

PCMCIA Programming • LAN NetView Development • New OS/2 Tools

November/December 1993

Display Until 12/31/93

os/2 Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

***SOMObject
Toolkit***

***Migrating to
Objects***

***Workplace Shell
Objects***

***IBM's Microkernel
Strategy***

Reusing Objects

**WRITING
OBJECT-ORIENTED
PROGRAMS**

U.S. \$9.95 Vol. 5 No. 5



0 74470 12531 0

WATCOM™

Visual Development Environment For

WATCOM VX•REXX is an easy to use visual development environment for creating applications that leverage the capabilities of OS/2 2.x and exploit the Presentation Manager graphical user interface. VX•REXX combines a project management facility, visual designer and an interactive source-level debugger to deliver a very approachable and highly productive visual development environment.

Design Applications Visually

Create rich graphical applications quickly and easily using the visual design environment. With the visual designer, you can graphically create Presentation Manager interface objects, quickly customize their properties, and easily attach REXX procedures using powerful drag-and-drop programming techniques.

Integrated Development Environment

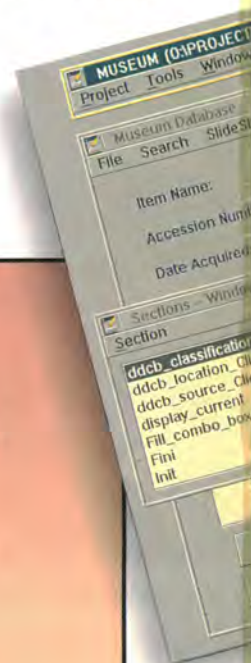
Build, test and debug your application without leaving the development environment. Then package your application as an EXE file or PM macro for royalty-free redistribution. The power of the integrated development environment and debugger can also be used with your existing REXX applications.

Powerful Open Environment

Enjoy the simplicity of event-driven programming together with the global editing capabilities essential for professional project management. WATCOM VX•REXX is open and extensible through IBM's object oriented System Object Model (SOM) technology. You can access all standard REXX API's including DB2/2, because VX•REXX is based on the OS/2 2.x standard system REXX.

Highlights

- ▶ Easy to use visual development environment
- ▶ Create and modify objects dynamically at both edit and run time
- ▶ Powerful project management facility
- ▶ Advanced interactive source-level debugger
- ▶ Package your applications as EXE files or PM macros
- ▶ Access to standard REXX API's including DB2/2
- ▶ System Object Model (SOM) based object manager
- ▶ Drag-and-drop programming
- ▶ Support for multi-threaded applications
- ▶ Include OS/2 style help and hints in your applications
- ▶ Supports SAA CUA'91 objects
- ▶ Integrated console window support for existing REXX programs
- ▶ Royalty-free run-time
- ▶ Multiple modeless window support
- ▶ Create PM macros for applications supporting REXX as a macro language



VX•REXX™

OS/2 REXX

Interactive Debugging

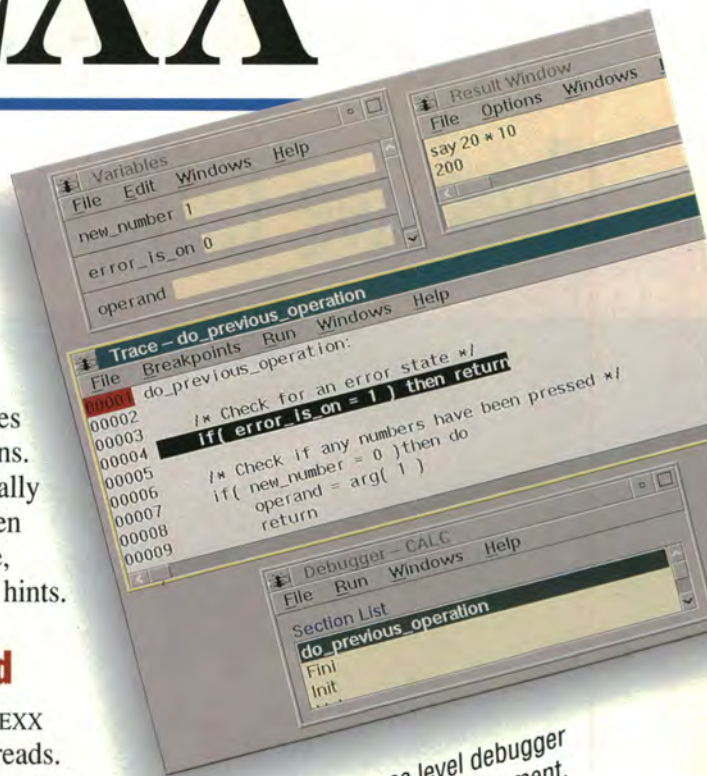
If an error occurs at run-time, VX•REXX will display a traceback pinpointing the source line where the error occurred. A simple click of the mouse will return you to the source edit window to correct the error. The built-in interactive source-level debugger lets you set breakpoints, step through code and watch variables to track down complex problems.

Build Professional Applications

WATCOM VX•REXX allows you to leverage key OS/2 features to create professional applications. Build applications that dynamically create and modify CUA'91 screen objects at both edit and run-time, and include OS/2 style help and hints.

Create Multi-Threaded Applications

Every VX•REXX application contains multiple threads. One thread remains responsive to user input while others continue processing. In addition, VX•REXX provides the ability for advanced applications to easily use additional threads.



The integrated source level debugger simplifies your project development.



WATCOM VX•REXX:

The integrated visual solution builder for OS/2 2.x.

Suggested Retail: \$199*

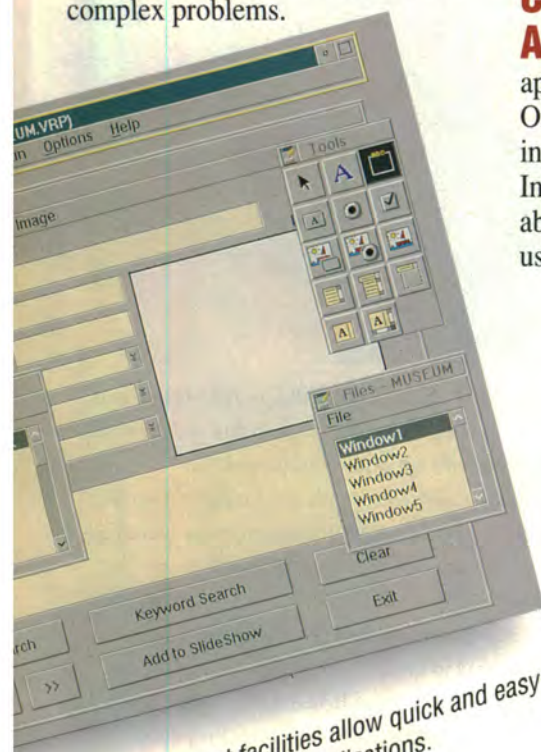
**Call Toll Free
1-800-265-4555**

WATCOM

WATCOM International

415 Phillip Street, Waterloo, Ontario, Canada, N2L 3X2
Phone: (519) 886-3700 Fax: (519) 747-4971

*Prices and specification are subject to change without notice. Price does not include freight and taxes where applicable. Prices quoted in US dollars. WATCOM, the Lightning Device, and VX•REXX are trademarks of WATCOM International Corporation. Other trademarks are the properties of their respective owners. © Copyright 1993 WATCOM International Corporation.



Project management facilities allow quick and easy development of your OS/2 applications.

“We were told it was impossible to develop a client/server application without extensive retraining. Then we talked to Micro Focus.”



Larry Lowder, Systems Architect, Questar Service Corporation.

Mountain Fuel Supply®, a division of Questar®, is a utility company supplying natural gas to 750,000 customers across Utah, Wyoming, Idaho and Colorado. The company's success is largely driven by its implicit belief that the customer is number one.

Yet, IT also plays its part in that success: client/server architectures and graphical user interfaces (GUIs) have helped Mountain Fuel Supply move applications and information closer to the customers and the employees. All of which has resulted in an augmented level of service being offered to customers.

When Larry Lowder, one of Questar's Systems Architects, set out to build the client/server architecture for Mountain Fuel Supply, he needed solutions, not skepticism. For the first project, a cashiering system, he needed to link workstations with OS/2® to the DB2® database on the host, running CICS®.

"We were faced with having to spend up to two years retraining our

COBOL programmers in C and API calls. Then we discovered Micro Focus Dialog System™. It allowed us to build the client functionality we required, and re-engineer the existing mainframe application as a server."

"Within a week, mainframe programmers were producing GUI screens for COBOL. Within 90 days we had delivered the system. Now we're not only coming in under budget, but also way ahead of schedule."

As Mountain Fuel Supply discovered, the Micro Focus solution lets you make a productive transition to client/server without sacrificing any of the resources you've worked so hard to build.

When the world's leading corporations demand "A Better Way of Programming,"™ they turn to Micro Focus. For a brochure on putting the Micro Focus Client/Server Solution to work for you, call 800-872-6265.

MICRO FOCUS®

Micro Focus Inc. 2465 East Bayshore Road, Palo Alto, CA 94303. Tel. (415) 856 4161.

OS/2 Developer

NOVEMBER/DECEMBER 1993

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT



EDITOR'S COMMENTS

On Sneakers, SOM, and Support 5



OBJECT-ORIENTED PROGRAMMING

SOMObjects Developer Toolkit: An Overview 6

Migrating from File-Based Storage to Object Database Storage On OS/2 38



GUI

An Object of Many Colors: Using Custom Controls Within a Workplace Object 46

Writing WPS-Compliant Applications 60



FUTURES

The IBM Microkernel Technology 70



DEVICES

Programming PCMCIA: It's All In The Cards 76



THE ADVANCED NOVICE

Reuse: Recognizing The Opportunities 82



LAN

Introduction To LAN NetView Application Development: An Xmp/Xom Primer 92



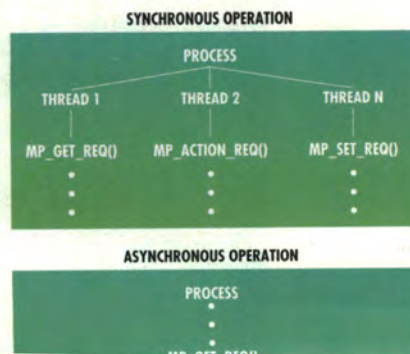
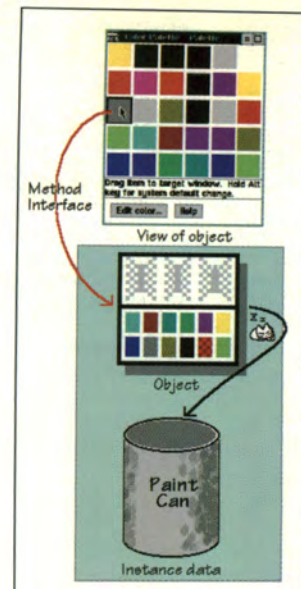
PRODUCT WATCH

New Tools, Applications, and Support 102



SOFTWARE TOOLS

Software Installer: Installations Made Easy 104





Programmer's Paradise®

Programmer's Paradise®
Italia
Phone: 39-2-480-16053
FAX: 39-2-480-16039

SourceLink v2.0 by One Up Corporation

SourceLink, the ultimate 32-Bit programming development tool, combines the functionality of hyperlink source code access, a fully-functional editor and extensive file manipulation utilities into one very powerful toolset. Now you can find code, change code, spawn compiles and hyperlink directly to the source of error quickly and efficiently. Features point & click source code navigation and automatic generation of function call trees.



List: \$299 Ours: \$259
FAXcetera #: 6005-0003

Introducing the Lotus SmartSuite 32-Bit Bundle for OS/2!

by Lotus Development Corp.

Lotus' new SmartSuite for OS/2 is the only complete 32-bit solution for OS/2 desktops. For a limited time, when you buy SmartSuite for OS/2 or a SmartSuite for OS/2 Upgrade*, you'll get OS/2 2.1 FREE. Act now to take advantage of this special introductory offer, while supplies last!

FREE
OS/2 2.1



	List: \$795	Ours: \$599
Upgrade*:	List: \$595	Ours: \$319
FAXcetera #:	4000-0027	



Window Washer v2.0 by One Up Corporation

The latest version of the best-selling 32-bit screen saver for OS/2, with full system password security and the most complete monitor burn-in protection available today. Version 2.0 features many exciting animated effects, digital video and audio (plays CD's, MIDI or WAV files with program effects). Also

utilizes TIF, GIF, BMP, & PCX backgrounds.

List: \$39.95 Ours: \$35
FAXcetera #: 6005-0001

Open Shutter v1.11 by One Up Corporation

Our easy-to-use screen capture process allows you to select any rectangular area, window, or entire desktop, and capture with a single user-defined keystroke or mouse click. Rotate, change colors, stretch/compress and then preview your modifications. Output to printer, clipboard, or soft copy in a variety of formats (BMP, ICO, TIFF, GIF, IMG, metafile, MacPaint).



List: \$69.95 Ours: \$59
FAXcetera #: 6005-0002

Gpf 2.1 Professional with Developer's Toolkit by Gpf Systems, Inc.

WYSIWYG GUI design environment. Full support for OS/2 2.x, OS/2 1.3.x, and MS Windows 3.0 and 3.1. Point and click to design a CUA '91 compliant GUI with no run time interpreter or royalties. Gpf generates C or C++ source, including SQL for DB2/2, as well as Help source, and all ancillary files (IPF/RTF, RC, h, DEF, MAK, etc.). All navigation and even user custom logic is added from within Gpf.



List: \$1440 Ours: \$1269
FAXcetera #: 3582-0002

SQL Objects++™ Database Class Library

SQL Objects++™ Database Class Library by Graphical Software Interfaces, Inc.

SQL Objects++™ Database Class Library is a powerful collection of object-oriented database classes that provide direct access to SQL and non-SQL databases. Each class is fully optimized to provide fast access to your data. No other access library is designed to fully utilize your OS/2 environment.

List: \$699 Ours: \$499
FAXcetera #: 1010-2601

Ami Pro 3.0 for OS/2 by Lotus Development Corp.

OS/2 users now have a native, 32-bit version of the award-winning Ami Pro 3.0 for Windows. Written from the ground up for OS/2, the new version of Ami Pro provides the highest level of word processing, networking and development power among OS/2 word processors.



	List: \$495	Ours: \$369
Upgrade:	List: \$129	Ours: \$ 99
FAXcetera #:	4000-0011	

GUARANTEED BEST PRICES! (Call for details)

To order call: 800-445-7899
Corporate (CORSOFT): 800 422-6507
FAX: 908 389-9227
International: 908 389-9228
Customer Service: 389-9229

For more information on the products
featured on these pages call—
FAXcetera®: (201) 762-1378

Programmer's Paradise®
1163 Shrewsbury Avenue
Shrewsbury, NJ 07702

• All prices are subject to change without notice.



EDITOR

Dick Conklin

EXECUTIVE EDITOR

Nicole Freeman

MANAGING EDITOR

Kelly A. Leighton

GROUP DIRECTOR

Don Pazour

PUBLISHER

Cathy Passage

ADVERTISING SALES

Holly Meintzer

(212) 626-2275

Michele Blake

(212) 626-2322

Dave Moreau

(212) 626-2318

Yvonne Labat

(415) 905-2353

MARKETING MANAGER

Susan McDonald

ART DIRECTOR/MARKETING

Christopher H. Clarke

ADVERTISING PRODUCTION**COORDINATOR**

Tom Marzella

DIRECTOR OF PRODUCTION

Andy Mickus

DIRECTOR OF CIRCULATION

Jerry Okabe

CIRCULATION DIRECTOR

John Rockwell

SUBSCRIPTION INQUIRIES

U.S.: (800) WANT-OS2

Foreign: (708) 647-5960



BY DICK CONKLIN

Editor's Comments

On Sneakers, SOM, and Support

UP AND RUNNING

Those of you who attended the Software Development '93 conference in Boston and were up and about at 6:30 in the morning may have noticed a strange sight. About 45 of our loyal



The Up and Running starting gate

readers participated in the first ever OS/2 Developer Up and Running 5K race.

The winner, Stephen Peckiconis of Lotus Corp., finished in 17 minutes and 17 seconds. He won a copy of Watcom's VX-REXX for OS/2.

Michelle Alvarez of Kodak was the first female finisher, with a time of 22 minutes and 26 seconds. She won two books from Van Nostrand Reinhold, Harkey and Orfali's client/server masterpiece and our own OS/2 2.X Notebook, both signed by the authors. (Michelle, was that a "photo finish?")

IBM VP John Patrick was the third IBM finisher, with a time of 25 minutes and 14 seconds. (Overall number 13—not bad for an IBM exec; where were Soyering, Reiswig, and Cannavino?)

YOUR HONOR, I OBJECT!

Sustained. OS/2 continues to prove itself as the operating system of choice for object programming. So we've called on SOM experts Roger Sessions, Nurcan Coskun, and Charles Erickson to share more

of their wisdom with us in this issue. They're joined by other object programmers who've contributed some great articles on various aspects of object-oriented programming.

UP AND COMING

In our January issue, we'll welcome back Brian Proffitt, who wrote the Software Tools column in the *Developer* for several years. Brian's new column will focus on the growing OS/2 corporate market as well as some of the development projects underway in places where once only mainframes feared to tread. If you have an interesting OS/2 application at your company, tell Brian about it on CompuServe at 75300,1466. And be sure to ask him about the book he co-authored, *OS/2 Inside and Out*.

Also joining us will be author (*Designing OS/2 Applications*) and long time OS/2 *Developer* contributor Dave Reich. Dave spends a lot of time supporting OS/2 developers, so he knows the answers to your most-often-asked questions. He also collects useful OS/2 tips and techniques and will bring us his choice Programming Pearls in each issue. Dave's Internet ID is speedracer@vnet.ibm.com.

Dick Conklin



Dick Conklin



IBM VP John Patrick, winner Stephen Peckiconis, Developer publisher Cathy Passage

IBM OS/2 Developer: Vol. 5, No. 5, November/December 1993

Subscription information: U.S.: One year (six issues), \$39.95. Canada, Mexico, and international surface mail, add \$16 per year for postage. Canadian GST no. 124513185. International air mail, add \$30 per year for postage. Foreign orders must be accompanied by payments in U.S. funds. For new orders and customer service, call (800) WANT-OS2 (800-926-8672) or (708) 647-5960 (fax: 708-647-0537). IBM employees and branch office customers can subscribe to *OS/2 Developer* through IBM Mechanicsburg's Systems Library Services (SLSS) using OS/2 Developer's order number: G362-0001. POSTMASTER: Please send address changes to OS/2 Developer, P.O. Box 1079, Skokie, IL 60076-8079. Allow eight weeks for subscriptions to begin. *OS/2 Developer* is published bimonthly by Miller Freeman Inc., 600 Harrison Street, San Francisco, CA 94107, (415) 905-2200. Application to mail at second class postage rates is pending at San Francisco, CA and additional mailing offices.

© Copyright 1993 by Miller Freeman Inc. PRINTED IN THE U.S.A.



Object-Oriented Programming

This article is an introduction to the SOMObjects Developer Toolkit 2.0 for OS/2 2.0 or 2.1. This major new release of the product, often described simply as "SOM," includes not only major upgrades on the SOM kernel itself but also includes several important frameworks that provide new development opportunities for the designer and implementor of SOM objects. BY ROGER SESSIONS, NURCAN COSKUN, and CHARLES ERICKSON

SOMObjects Developer Toolkit: An Overview



Roger Sessions



Nurcan Coskun



Charles Erickson

The SOMObjects Toolkit is a product designed for programmers involved in object-oriented programming. The Toolkit includes both the SOM Kernel and the SOM Frameworks. The SOM Kernel provides a mechanism for writing language-independent object-oriented classes. By language-independent, we mean that the class consumer can use a different language than that used by the class implementor, as long as both consumer and implementor languages supports SOM bindings. The SOM Frameworks greatly decrease the programming effort required to give those classes basic functionalities such as persistence and object distribution.

In this article, we'll introduce you to the use of the SOM Kernel and SOM compiler. These two components provide the basic functionality needed to create classes. We'll develop a complete example and give you all the information you'll need to compile and run each example. We assume only that you have installed the toolkit according to the installation directions. This will allow you to use the examples as the basis for your own programming efforts.

INTRODUCTION TO THE SOMOBJECTS TOOLKIT

The process of creating a SOM class is summarized in Figure 1. The programmer defines a class in a language known as the Interface Definition Language, or IDL. This definition is placed in a file typically named with the extension `.idl`. The SOM compiler is run against this IDL file and three files

are generated. The first is a header file used for class implementation code, named with the extension `.ih`. The second, a header file used by clients, is named with the extension `.h`. The third is a method template file containing C code skeletons for each method described in the class definition, named with the extension `.c`. These skeletons are filled in by the programmer.

Programmers who used the first version of SOM will notice right away that the class definition language has changed. Classes used to be defined with a language known as OIDL, or Object Interface Definition Language. Although OIDL is still supported, we are encouraging people to port their code to the IDL syntax for several reasons. Perhaps the most important is that the IDL syntax used to define classes and the associated C bindings has been accepted as an industry standard. The standard has been worked out through the Object Management Group (OMG) and their CORBA specification. Another reason for moving to IDL is that it allows use of new SOM Frameworks including Distributed SOM or DSOM.

As changes are made to the IDL file, the SOM compiler must be rerun. The `.ih` and `.h` files are regenerated and the `.c` file modified to add new skeletons for any newly defined methods. Existing method code is left untouched.

BASIC PROGRAMMING EXAMPLE

Let's start by looking at the definition, implementation, and use of a simple SOM class: an appointment. An object of the appointment class knows



about a particular appointment during the day. Appointments are characterized by a date, starting time, ending time, and subject. The subject is a description of the appointment's purpose.

A high-level description of this class in class definition pseudocode is shown in Figure 2. `Appointment` is derived from `SOMObject`, the default root class for all class derivations. We have indicated the direction of all parameters. The keyword `in` is used to indicate a parameter that is not updated. All of our parameters are `in`.

There are a series of relationships between certain instance data and pairs of methods. For example, there are the instance data `year` and the two methods `set_year()` and `get_year()`. It is quite common to have a particular piece of instance data associated with a set method, which sets the value, and a get method, which returns the value of the instance data.

There is a shortcut to declaring an instance data, a set and a get method—declaring an attribute. If we see an attribute `year`, we can interpret this as declaring a data element `year`, and a set and a get method. By using attributes, we can simplify Figure 2. The new high-level description of `Appointment` using attributes is shown in Figure 3.

`Appointment` includes a method named `bufferize()`. We will follow a rule that says object implementation and I/O should be kept separate. Another way of saying this is that objects should be separated from code that prints them out or reads them in. We use the method `bufferize()` as a convenient way to request an `Appointment` object to return a formatted string information about its contents. The separation of object implementation from I/O will greatly simplify the use of several frameworks of the `SOMObjects Toolkit`.

We also want `Appointment` to override certain methods inherited from `SOMObject`. The overridden methods are `somInit()`, the method invoked on an object when that object is first created, and `somUninit()`, the method invoked on an object when that object is destroyed. Updating Figure 3 to show overridden methods gives Figure 4.

Notice that `SOM` has predeclared certain types. These types can be used in the declaration of attributes, return values for methods, and parameters of methods. The valid types are shown in Figure 5. In this article, we use only `short` (which is

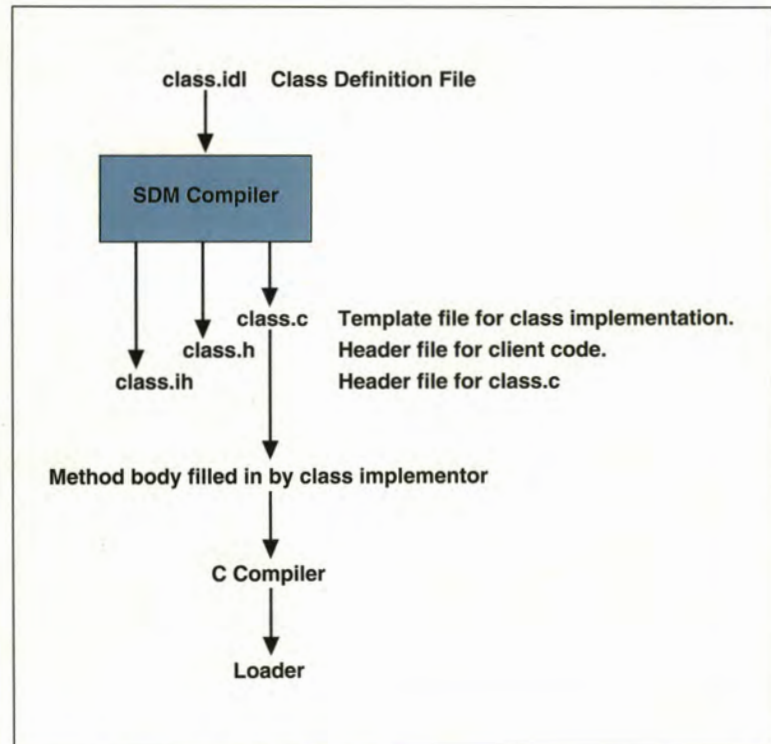


Figure 1. Summary of the SOM process

```
class: Appointment
derived from: SOMObject
instance data: short year
               short month
               short day
               short startingTime
               short endingTime
               string subject
               short id
methods: void set_year(in short year)
         short get_year()
         void set_month(in short month)
         short get_month()
         void set_day(in short day)
         short get_day()
         void set_startingTime(in short startingTime)
         short get_startingTime()
         void set_subject(in string subject)
         string get_subject()
         string bufferize(in string buffer)
         short get_id()
         void set_id(in short id)
```

Figure 2. High-level description of `Appointment` class



equivalent to the C type `short int`), `string` (equivalent to the C type `char*`), and `sequence` (equivalent to a specific structure in C).

The IDL file that defines the `Appointment` class, `appt.idl`, is shown in Figure 6. Line numbers are given for ease of reference. It should be relatively easy to relate this back to the high level description shown in Figure 4.

Let's go through the IDL file shown in Figure 6 line by line.

Lines 01, 02, and 52 ensure that this file is compiled only once. This is similar to the technique used in C++ to ensure that class definitions in that language are compiled only once (see Sessions, *Class Construction in C and C++*).

Line 04 includes the IDL definition for the base class of this class.

Line 06 declares `Appointment` to be an interface (or a class, for our purposes) and notes that the class is derived from `SOMObject`. Notice that the interface declaration encompasses lines 07-50, inclusive, as indicated by the brackets.

Lines 12, 15, 18, 21, 24, 27, and 30 declare the attributes for the class. Line 08 declares the only method of the class.

The implementation section of the class declaration begins at line 32 and ends at line 49 (again as indicated by brackets). This section contains information not considered part of the interface per se, but still necessary to the class declaration. Pragmatically, the implementation section contains any information that is an extension of the OMG CORBA standard. Notice that the implementation section is within the interface section.

Within the implementation section, we see a `dllname` statement at line 34. The purpose of this is to tell SOM the name of the dll into which the object code for this class will be built. This information is used by SOM to locate and dynamically load class libraries.

Also within the implementation section, we see a `releaseorder` section from lines 35-38. This section is designed to ensure that base classes can be modified without forcing derived classes to be recompiled. We will not explain

```
class: Appointment
derived from: SOMObject
attributes: short year
           short month
           short day
           short startingTime
           short endingTime
           string subject
methods: string bufferize(in string buffer)
```

Figure 3. High-level description of `Appointment` class using attribute

```
class: Appointment
derived from: SOMObject
attributes: short year
           short month
           short day
           short startingTime
           short endingTime
           string subject
methods: string bufferize(in string buffer)
overridden methods: somInit
                  somUninit
```

Figure 4. High-level description of `Appointment` class using attributes and showing overridden methods

Basic Types	Constructed Types	Template Types
short	struct	sequence
long	union	string
unsigned short	enum	
unsigned long		
float		
double		
char		
octet		
boolean		
any		

Figure 5. Valid IDL types

how this works technically, but the rules for implementing this section are: Whenever a method is added to the class, it is added to the release order. Methods should never be removed from the release order (even when

removed from the class) and methods should never be moved within the release order (even when moved within the class). Notice that these rules apply to methods implied by attributes, (such as `_get_year()`) as well as explicit



```
01 #ifndef appt_idl
02 #define appt_idl
03
04 #include <somobj.idl>
05
06 interface Appointment : SOMObject
07 {
08     string bufferize();
09 // Returns the external representation of the appointment
10 // as a string.
11
12     attribute short year;
13 // The year of the appointment.
14
15     attribute short month;
16 // The month of the appointment.
17
18     attribute short day;
19 // The day of the appointment.
20
21     attribute short start;
22 // The starting time of the appointment.
23
24     attribute short end;
25 // The ending time of the appointment.
26
27     attribute string subject;
28 // The subject of the appointment.
29
30     attribute short apptId;
31
32     implementation {
33
34         dllname = "apptbk.dll";
35         releaseorder: bufferize, _get_year, _set_year,
36             _get_month, _set_month, _get_day, _set_day,
37             _get_start, _set_start, _get_end, _set_end,
38             _get_subject, _set_subject, _get_apptId, _set_apptId;
39
40 // Method Modifiers
41     somInit: override;
42     somUninit: override;
43
44 // Attribute Modifiers
45     subject: noset;
46
47 // Instance data
48     string bp; // buffer used by bufferize method
49 };
50 };
51
52 #endif /* appt_idl */
```

Figure 6. IDL definition for *Appointment*, *appt.idl*

"No more



Gary Slattery, Software Developer, Computer Associates

"The best platform for DOS and Windows."

"I develop software applications for a living and I think OS/2® is a great way to do business." A 32-bit, virtual memory operating system, OS/2 is the ideal platform for developing your DOS, OS/2, Windows™ and even host-based applications.

With OS/2 you can boost the power of your favorite DOS and Windows tools,

plus take advantage of over 250 available OS/2 development tools and utilities.

"It's a productivity thing."

"I use CA Realizer to prototype new graphical interfaces. I write code using CA C++™ and CommonView™ (integrated with IBM's C Set/2™ compiler) while I compile in the background. And when designing reports or planning schedules, I use CA-RET."

OS/2's pre-emptive multithreaded multitasking dynamically manages CPU time so you can run DOS, Windows and OS/2 apps concurrently in different sessions with maximum efficiency. That means you can edit in one window, compile in another, link in a third and test in a fourth. With OS/2 Crash Protection™, if one application goes down due to a bug, the rest you're working on won't. "There's no limit to what you can do with this system.... It's definitely made me more productive."



The Development Platform of Choice

- Enhanced development platform for DOS, Windows and OS/2 apps.
- OS/2 Crash Protection for superior reliability.
- Pre-emptive multitasking for increased productivity.
- Virtual memory provides up to 512MB per session.
- Flat memory model eliminates wrestling with segments.
- Multiple virtual DOS machines for concurrent app testing.
- Object-oriented Workplace Shell is easy and intuitive.



arrested development."

The object-oriented user interface—the Workplace Shell™ (WPS)—gives you easy control with direct manipulation of visual objects on your computer screen. And should you need assistance with anything, IBM's Worldwide Developer Assistance Program is always there to help.

"A developer's dream."

With OS/2's 32-bit architecture, you can push your 386 and 486 hardware to the limit, and develop spectacular multimedia and enterprise-wide client-server applications. OS/2 lets you create the widest range of applications, for a variety of platforms, for almost any size computer. Here's your chance to develop truly revolutionary 32-bit applications. With OS/2, now there's nothing stopping you.

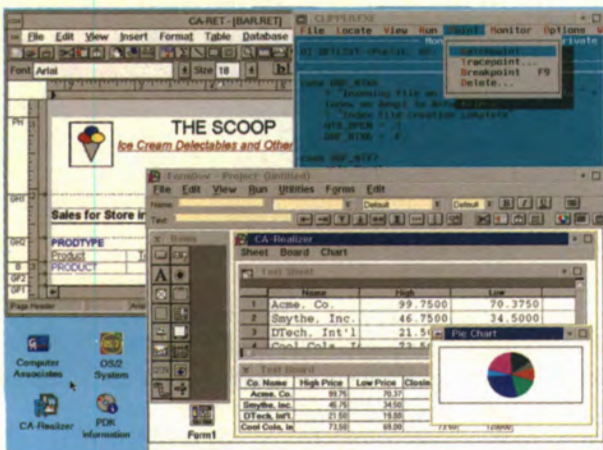
To learn more, call 1 407 982-6408 now, and get a free white paper on why OS/2 is the ideal platform for your development efforts.

Operate at a higher level.

1992
- WINNER -
PC Magazine Award
for Technical Excellence



OPERATING SYSTEMS AND
SOFTWARE STANDARDS
OS/2, Version 2.0
IBM Corporation



Use OS/2 to increase productivity of DOS, Windows and OS/2 application development.





methods (such as `bufferize()`).

The overrides are declared as shown in lines 41 and 42. Notice that parameters of overridden methods are not given here; they can be inferred from the IDL file describing the base class.

Attributes can be modified, as shown in line 45. The only attribute modifier we use is `noset`. The keyword `noset` tells the SOM compiler that we don't want a set method automatically generated for this attribute. Instead, we want to write our own. We'd do this in a situation where the set method that would have been generated wouldn't be adequate for our purpose. The default set methods generated for string attributes are almost always inappropriate and must be written by hand.

We can see why it is best to hand code the set methods for string attributes if we look at what would have been generated for the set method had we not used `noset`. The default set method is shown in Figure 7.

The default set in Figure 7 is rarely appropriate for any attribute that is essentially a pointer to something because it results in memory leakage. A better algorithm for a set involving a pointer looks like:

- See if data member is already pointing to something
- If so, free what it is pointing at
- Allocate memory for the new value
- Copy the parameter to the newly allocated memory
- Set the data member to the newly allocated memory

By using the `noset` modifier in line 45 of Figure 6, we reserve for ourselves the right to write our own set for the attribute `subject`.

Having completed the definition of `Appointment`, we are ready to use the SOM compiler to generate a file containing method skeletons. The command to accomplish this is:

```
sc -sc appt
```

where `appt` is the root name of the file containing the definition of `Appointment`, and the `.idl` extension is assumed.

This command produces the `.c` file

```
SOM_Scope void SOMLINK _set_subject(Appointment *somSelf, Environment *ev,
string subject)
{
    AppointmentData *somThis = AppointmentGetData(somSelf);
    AppointmentMethodDebug("Appointment", "_set_subject");

    somThis->subject = subject;
}
```

Figure 7. Default set method for attribute `subject`

```
#define Appointment_Class_Source
#include <appt.ih>

SOM_Scope string SOMLINK bufferize(Appointment somSelf, Environment
*ev)
{
    AppointmentData *somThis = AppointmentGetData(somSelf);
    AppointmentMethodDebug("Appointment", "bufferize");
    return;
}

SOM_Scope void SOMLINK _set_subject(Appointment somSelf, Environment *ev,
string subject)
{
    AppointmentData *somThis = AppointmentGetData(somSelf);
    AppointmentMethodDebug("Appointment", "_set_subject");
}

SOM_Scope void SOMLINK somInit(Appointment somSelf)
{
    AppointmentData *somThis = AppointmentGetData(somSelf);
    AppointmentMethodDebug("Appointment", "somInit");
    Appointment_parent_SOMObject_somInit(somSelf);
}

SOM_Scope void SOMLINK somUninit(Appointment somSelf)
{
    AppointmentData *somThis = AppointmentGetData(somSelf);
    AppointmentMethodDebug("Appointment", "somUninit");

    Appointment_parent_SOMObject_somUninit(somSelf);
}
```

Figure 8. SOM Compiler generated `.c` file from `appt.idl`

shown in Figure 8.

Our job is to take the skeleton file in Figure 8 and fill in the body of the methods. Notice that the default `set` and `get` methods do not appear in this file, although those identified by `noset`

do appear. (The default `set` and `get` methods can be found in the SOM compiler generated `appt.ih` file.) The filled-in skeleton file is shown in Figure 9, again with line numbers added to facilitate discussion.

C Set++

Get set for incredible 32-bit power. Get set for mission critical reliability. Get set for a range of advanced features. Get C Set++™ from IBM Programming Systems. C Set++ is the most complete object-oriented application development package you can buy for OS/2®.

C Set++ lets you create the most advanced, high-performance applications imaginable. Its 32-bit C/C++ compiler lets you unleash all the power of OS/2, giving you industrial-strength code for your mission critical applications. It has an extraordinary code optimizer with a full set of options—even a switch to optimize for the new Pentium™ processor. Plus there's a full set of class libraries, including application

for the

mission

frameworks for PM, container classes and classes for multitasking, streams and more.

There's a whole set of other helpful features, like an interactive source level debugger. The unique Execution Trace Analyzer traces the execution of a program, then graphically displays diagrams of the analysis. You also get Workframe/2™, a language-independent tool that lets you customize your own environment. It's adaptable and flexible—you can use any 16 and 32-bit DOS, Windows™ and OS/2 tools.

critical set.



Set your eyes on this: C Set++ gives the Workplace Shell™ quite a workout.

With C Set++, it's easier than ever to set your sights on success. To order or to find out more about OS/2 2.1 or C Set++, call 1 800 3-IBM-OS2. In Canada, call 1 800 465-7999, ext. 460.

Operate at a higher level.™

IBM and OS/2 are registered trademarks and C Set++, Workplace Shell, Workframe/2 and "Operate at a higher level" are trademarks of International Business Machines Corporation. Pentium is a trademark of Intel Corporation. Windows is a trademark of Microsoft Corp. ©1993 IBM Corp.



Let's go through some of the more interesting lines in the filled-in skeleton in Figure 9. First we see some bookkeeping lines (lines 1-2). These are added automatically by the SOM compiler, and we will not discuss their purpose here. Similarly, the keywords `SOM_Scope` and `SOMLINK` are seen within the method declarations. They are typically equated to `static` and the null string respectively. Therefore the line:

```
SOM_Scope string SOMLINK bufferize(
```

can be thought of as being equivalent to:

```
static string bufferize(
```

We will not go into the reason for `SOM_Scope` and `SOMLINK`.

The SOM compiler has automatically supplied the correct parameters for each method, and that in addition to the parameters (if any) specified by the developer, SOM has supplied two implied parameters to every method (see, for example, line 5.) The first implied parameter is the receiving object, or what some authors describe as the target object. If you are not familiar with the concept of a target or receiving object, refer to *Class Construction in C and C+* (see the Bibliography). The second implied parameter is the `Environment` variable. Its primary purpose is to relay error conditions, a topic we will cover in a later article.

Each method starts with some more bookkeeping lines such as 18 and 19, which we will not discuss here. If you need to declare local variables, they must precede the SOM supplied debug line. (For example, if the method `somInit()` needed to declare local variables, they would go before line 34.)

When a method needs to refer to a data element, even when the data element is implied by an attribute, we refer to it using an underscore, as in line 23:

```
_subject = NULL;
```

```
01 #define Appointment_Class_Source
02 #include <appt.ih>
03 #define BUF_SIZE 512
04
05 SOM_Scope string SOMLINK bufferize(Appointment somSelf, Environment *ev)
06 {
07     AppointmentData *somThis = AppointmentGetData(somSelf);
08     AppointmentMethodDebug("Appointment", "bufferize");
09
10     sprintf(_bp, "(%d) Date: %d/%d/%d Start: %d End: %d\nSubject      : %s\n",
11         _apptId, _month, _day, _year, _start, _end, _subject);
12     return(_bp);
13 }
14
15 SOM_Scope void SOMLINK _set_subject(
16     Appointment somSelf, Environment *ev, string subject)
17 {
18     AppointmentData *somThis = AppointmentGetData(somSelf);
19     AppointmentMethodDebug("Appointment", "_set_subject");
20
21     if (_subject != NULL) {
22         SOMFree(_subject);
23         _subject = NULL;
24     }
25     if(subject == NULL) return;
26     _subject = SOMMalloc(strlen(subject)+1);
27     strcpy(_subject, subject);
28 }
29
30 SOM_Scope void SOMLINK somInit(
31     Appointment somSelf)
32 {
33     AppointmentData *somThis = AppointmentGetData(somSelf);
34     AppointmentMethodDebug("Appointment", "somInit");
35
36     Appointment_parent_SOMObject_somInit(somSelf);
37     _year = 0;
38     _month = 0;
39     _day = 0;
40     _start = 0;
41     _end = 0;
42     _subject = NULL;
43     _apptId = 0;
44     _bp = SOMMalloc(BUF_SIZE);
45 }
46
47 SOM_Scope void SOMLINK somUninit(
48     Appointment somSelf)
49 {
50     AppointmentData *somThis = AppointmentGetData(somSelf);
51     AppointmentMethodDebug("Appointment", "somUninit");
```

Figure 9. Filled in skeleton file for *Appointment* (continued on page 16)

Where

This is the program, and the platform, BASIC was built for. Together, CA-REALIZER® 2.0 and OS/2® 2.1 go above and beyond BASIC.

With CA-REALIZER, you get a visual program-

CA-REALIZER

is BASIC-ally

ming tool that lets you develop visual applications with maximum impact.

It's a structured superset of BASIC, extended to access objects and resources of both Windows™ and OS/2.

You also get fully integrated Programmable Application

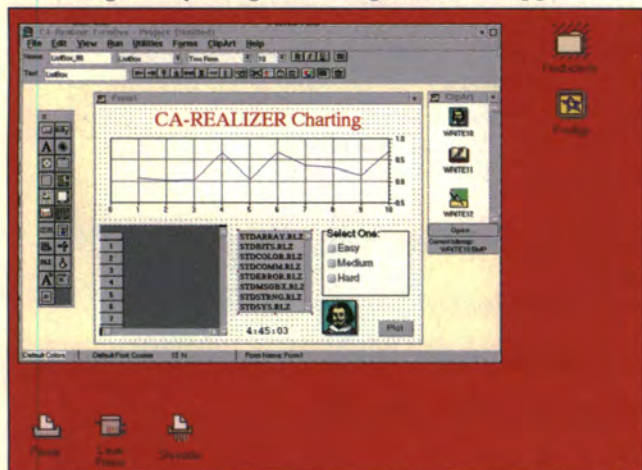
Tools.™ In short, it's a developer's dreamkit.

CA-REALIZER eliminates the need for cumbersome SDKs. A totally new and improved visual FormDev lets you edit multiple forms simultaneously, as well as adding items like scroll bars, spreadsheets, charts, animation and OLE objects. You can even import or export 1-2-3, Excel and Xbase files.

With its pre-emptive multitasking, OS/2 makes CA-REALIZER a faster BASIC to work with, allowing you to move on while the system completes the previous job. And the applications you create can be compiled into stand-alone OS/2 programs you can distribute royalty-free with the run-time module that's included. There's even an award-winning report writer at no extra charge.

Clearly, when it comes to BASIC, CA-REALIZER is anything but. To order or find out more about OS/2 2.1 or CA-REALIZER, call 1 800 3-IBM-OS2*. In Canada, call 1 800 465-7999.

Operate at a higher level. better.



Realize the full power of your PC with OS/2, and you'll never go back to BASICs.



*Or call Computer Associates at 1 800 CALL-CAI. IBM and OS/2 are registered trademarks and "Operate at a higher level" is a trademark of International Business Machines Corporation. CA-REALIZER is a registered trademark of Computer Associates International, Inc. Windows is a trademark of Microsoft Corporation. All other products are trademarks or registered trademarks of their respective companies. ©1993 IBM Corp.

The IBM logo, consisting of the letters "IBM" in a stylized, bold, sans-serif font.



The underscore is used to indicate a data element of the target object of this method invocation.

Memory allocation and deallocation uses the SOM-supplied functions `SOMMalloc()` and `SOMFree()` (as seen in lines 26 and 59) instead of their standard C counterparts.

When you want to invoke the method defined for your base class, use the syntax `MyClassName_parent_BaseClassName_MethodName` (parameters). A typical example of this is seen in line 36, where we have the `somInit()` method invoking its base classes `somInit()`. `somInit()` is the method automatically invoked on an object when that object is first created. This is where one typically places any initialization code.

The implementation file includes code for `set_subject()`, the set method for the attribute `subject` that we wanted to write ourselves via the `noset` modifier. The code for this method follows the algorithm given earlier for set methods involving pointer attributes.

The initialization method, `somInit()`, sets up a newly instantiated `Appointment` object. Typically the first thing an implementation of `somInit()` does is to request the base class initialization, thus the invocation in line 36. The deinitialization method, `somUninit()`, performs the reverse operation.

A very simple client program that uses this class is shown in Figure 10. Notice that the client code has no knowledge of the `Appointment` class other than the interface section of the IDL.

If we look at the client program shown in Figure 10, the first thing we see is the use of the SOM compiler generated `appt.h` file. This file contains all of the information needed for a client to use a SOM class.

The client declares an object of class `Appointment` and instantiates the object, as shown in line 5. The client invokes methods using an underscore preceding method names (as shown in line 17) and a double underscore preceding set or get methods for attributes (as shown in lines 8-14).

All methods take as their first para-

```
52
53     _year = 0;
54     _month = 0;
55     _day = 0;
56     _start = 0;
57     _end = 0;
58     if (_subject != NULL) {
59         SOMFree(_subject);
60         _subject = NULL;
61     }
62     SOMFree(_bp);
63     Appointment_parent_SOMObject_somUninit(somSelf);
64 }
```

Figure 9. Filled in skeleton file for `Appointment` (continued from page 14)

```
01 #include <appt.h>
02
03 main ()
04 {
05     Appointment appt = AppointmentNew();
06     Environment *ev = SOM_CreateLocalEnvironment();
07
08     __set_month(appt, ev, 12);
09     __set_day(appt, ev, 05);
10     __set_year(appt, ev, 1993);
11     __set_start(appt, ev, 900);
12     __set_end(appt, ev, 1100);
13     __set_subject(appt, ev, "Product Planning");
14     __set_apptId(appt, ev, 1);
15
16     somPrintf("%s\n", _bufferize(appt, ev));
17     _somFree(appt);
18     SOM_DestroyLocalEnvironment(ev);
19 }
```

Figure 10. Sample client program For `Appointment` class

```
(1) Date: 12/5/1993 Start: 900 End: 1100
Subject      : Product Planning
```

Figure 11. Output from the sample client program

meter the receiving object. Most methods take as their second parameter a pointer to an environment. (This will be discussed in a later article on error handling. For now, declare and allocate an environment using the syntax shown in

line 6 and free the associated memory using the syntax shown in line 18.)

Notice that not all methods take an environment pointer. Assuming you use standard IDL processing, any methods you declare will take an envi-

This VX•REXX

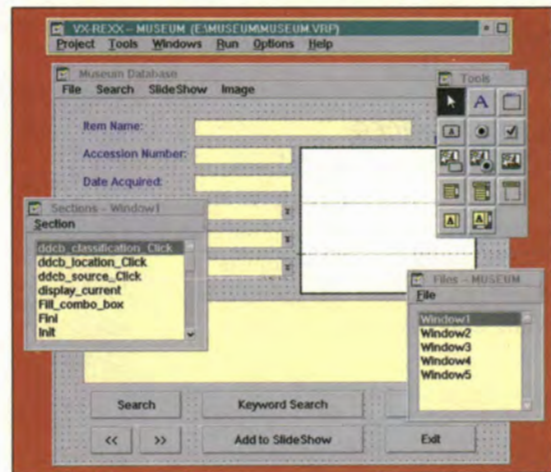


Developers: on your marks, get set—GUI!
With VX•REXX,™ WATCOM™'s
visual development environment
for OS/2® REXX, you're on the fast
track to creating applications that
exploit the graphical user interface
capabilities of OS/2 and the Workplace Shell.™

will really

 VX•REXX is an easy to use, powerful and fully
integrated environment that combines a project
management facility, visual designer and
an interactive source-level debugger to
deliver a very approachable and highly
productive visual development environ-
ment. With the visual designer, you can graphically
create CUA '91 objects, quickly customize their

get you



*With the visual development environment of VX•REXX,
it's all systems GUI.*

properties, and easily attach REXX proce-
dures to the objects.

Since it's based on IBM System Object
Model (SOM) technology, the VX•REXX
environment is wide open. You can access
all standard REXX API's including DB2/2. VX•REXX
also lets you package applications as EXE files or PM
macros. There's support for multithreaded applica-
tions. You can even include OS/2 style help and hints
in your applications.

To keep GUI-ing, WATCOM customer support
delivers timely response by phone, fax, e-mail,
CompuServe,™ or the WATCOM bulletin board

GUI-ing.

system. So when it comes to developing
OS/2 applications, VX•REXX is the way
to GUI. To order or to find out more about OS/2 2.1
or VX•REXX, call 1 800 3-IBM-OS2. In Canada,
call 1 800 465-7999.

Operate at a higher level.™



ronment pointer. The only methods you are likely to run into that do not take an environment pointer are those declared in `SOMObject` and `SOMClass`.

We recommend, but do not require, that you use `somPrintf()` instead of `printf()` for output as shown in line 16. This ensures that any character redirection will be properly handled across the board.

Good programming manners requires that you free all of your objects at the end of your program. This is accomplished through the use of `_somFree()`, as shown in line 18. Invoking `_somFree()` on an object automatically invokes the `somUninit()` method that we overrode for this class. The code for the overridden `somUninit()` is shown starting at line 47 in Figure 9.

Running this program gives the output shown in Figure 11.

A MORE ADVANCED PROGRAMMING EXAMPLE

Now that we've created the `Appointment` class, we'll use it as a basis for several other classes. Recall that objects of `Appointment` have a date, a starting time, an ending time, a subject, and an ID.

In this section we will derive two new classes from `Appointment`. The first derived class is `Meeting`, which has a `location` in addition to what it inherits from `Appointment`. The second derived class, `ConferenceCall`, contains a `phoneNumber`.

We will then create a constructed class, `AppointmentBook`, which will contain a sequence of `Appointments`. When we have a sequence of objects of class `Appointment`, by the rules of inheritance any of the sequence can be either an object of class `Appointment` or some class derived from `Appointment`. Since both `ConferenceCall` and `Meeting` are derived from `Appointment`, any of these three classes can be an element in the `AppointmentBook`.

In the process of discussing `AppointmentBook`, we will discuss one of the template types shown in Figure 5, `sequence`.

The IDL file defining the `Meeting` class, `mtg.idl`, is shown in Figure 12. There are several points worth men-

```
01 #ifndef mtg_idl
02 #define mtg_idl
03
04 #include "appt.idl"
05
06 interface Meeting : Appointment
07 {
08     attribute string location;
09     // The location of the appointment.
10
11     implementation {
12
13         dllname = "apptbk.dll";
14         releaseorder: _get_location, _set_location;
15
16         // Method Modifiers
17         somInit: override;
18         somUninit: override;
19         bufferize: override;
20
21         // Attribute Modifiers
22         location : noset;
23
24         // Instance data
25         string bp; // buffer used by bufferize method
26     };
27 };
28
29 #endif /* mtg_idl */
```

Figure 12. IDL file for meeting: `mtg.idl`

tioning about this file. This class is derived from `Appointment` (see line 6), whereas `Appointment` was derived from `SOMObject`. The object code for this class will be stored in the same DLL file as the code for `Appointment`, since they both use the same `dllname` clause (see line 13). The `Meeting` class adds one new attribute to those it inherits from `Appointment`; that attribute is `location` (see line 8). One new attribute yields two implied methods, a `set` and a `get`, both reflected in the release order (see line 14). Since this attribute is a string, we want to write our own `set` method, as we did for the string attributes of `Appointment`. Thus we declare the attribute `noset` (see line 22). Because we also want to override the `bufferize()` method, we declare this

method as `override` (see line 19). On line 25 we have `bp` as the internal instance data to be used by the `bufferize` method.

Figure 13 shows the method implementation file for `Meeting`. As with `Appointment`, most of the file was automatically generated by the SOM compiler with the command:

```
sc -sc mtg
```

The methods are very similar to the methods of `Appointment`, and require no further discussion.

Figure 14 shows the IDL definition for the `ConferenceCall` class. The only substantive difference between this and the definition for `Meeting` is that now we

Welcome

Everyone knows OS/2® gives you more applications to choose from. Now all you need is more space on your hard disk to load them all. Stacker® for OS/2 & DOS is your answer. Stacker lets you quickly and safely double the capacity of your FAT hard disk, so you can take full advantage of OS/2—and virtually any DOS, Windows™ and OS/2 application you want.



You can install Stacker in minutes. With Express or Custom Setup, you can automatically compress the data on the drive or customize the Stacker configuration to fit your needs. And you can be sure your data is safe with the leader in disk expansion technology. In fact, over four million people worldwide already trust their data to patented Stacker LZS™ compression technology.

to the the expanding world



With Stacker, doubling the capacity of your hard disk is as easy as pie.

Stacker fully supports Boot Manager and Dual Boot configurations. And there's full support for OS/2's extended

attributes, too. The AutoProtect™ feature detects disk errors at boot-up and immediately protects your Stacker volume. AutoRecovery™ automatically repairs errors on the disk. And Stacker Optimizer™ even defragments a Stacker drive for optimal performance.

With Stacker, you can get the most from your FAT hard drive. And with OS/2 on it, you can get the most from your computer. To order or to find out more about OS/2 2.1 or Stacker for OS/2 & DOS, call 1 800 3-IBM-OS2. In Canada, call 1 800 465-7999.

Operate at a higher level. of Stacker.

IBM and OS/2 are registered trademarks and "Operate at a higher level" is a trademark of International Business Machines Corporation. Stac and Stacker are registered trademarks and LZS, AutoProtect, AutoRecovery and Stacker Optimizer are trademarks of Stac Electronics. Windows is a trademark of Microsoft Corporation. © 1993 IBM Corp.




```

#define Meeting_Class_Source
#include <mtg.ih>

#define BUF_SIZE 512

SOM_Scope void SOMLINK _set_location(Meeting somSelf, Environment *ev,
                                     string location)
{
    MeetingData *somThis = MeetingGetData(somSelf);
    MeetingMethodDebug("Meeting", "_set_location");

    if (_location != NULL) {
        SOMFree(_location);
        _location = NULL;
    }
    if (location == NULL) return;
    _location = SOMMalloc(strlen(location)+1);
    strcpy(_location, location);
}

SOM_Scope void SOMLINK somInit(Meeting somSelf)
{
    MeetingData *somThis = MeetingGetData(somSelf);
    MeetingMethodDebug("Meeting", "somInit");

    Meeting_parent_Appointment_somInit(somSelf);
    _location = NULL;
    _bp = SOMMalloc(BUF_SIZE);
}

SOM_Scope void SOMLINK somUninit(Meeting somSelf)
{
    MeetingData *somThis = MeetingGetData(somSelf);
    MeetingMethodDebug("Meeting", "somUninit");

    if (_location != NULL) {
        SOMFree(_location);
        _location = NULL;
    }
    SOMFree(_bp);
    Meeting_parent_Appointment_somUninit(somSelf);
}

SOM_Scope string SOMLINK bufferize(Meeting somSelf, Environment *ev)
{
    string pbp;
    MeetingData *somThis = MeetingGetData(somSelf);
    MeetingMethodDebug("Meeting", "bufferize");

    pbp = Meeting_parent_Appointment_bufferize(somSelf, ev);
    _bp = SOMRealloc(_bp, BUF_SIZE + strlen(pbp));
    sprintf(_bp, "%slocation : %s\n", pbp, _location);
    return(_bp);
}

```

Figure 13. C code for meeting: mtg.c

Catch

This just in. Applications Manager, best known as *AM*,™ has just added OS/2® 2.1 support. We repeat, the flagship client/server application development software from Intelligent Environments is now available for OS/2 2.1 environments.



For the latest-breaking developments, we take you to corporate America, where *AM* and OS/2 offer a tried and tested mechanism for building 32-bit, multitasking, line-of-business client/server applications. *AM*'s visual programming environment streamlines the development and maintenance of mission-critical client/server applications by teams of programmers. And Static SQL support makes *AM* a real headliner.

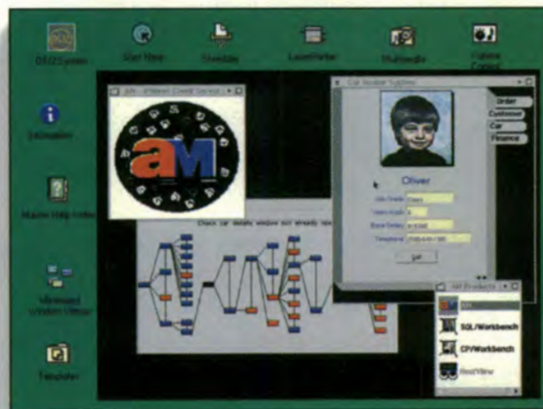
This recent news is becoming quite a feature story. By interfacing with MPPM/2, included with OS/2 2.1, *AM* lets programmers employ innovative team development capabilities

AM

like dynamically linked programming—simplifying reuse and maintenance of program code. DDE support allows programmers to paste information, including *AM* code, comments and *AM*-generated documentation, into Windows™ 3.1 applications easily. And there's also quicker screen interaction and improved productivity through support for OS/2's new high-performance 32-bit graphics engine.

With *AM* and OS/2, you'll never again have to return to your regularly scheduled programming.

the latest



Use *AM* to develop your own highly rated network programs.

news.

To order or to find out more about OS/2 2.1 or *AM*, call 1 800 3-IBM-OS2. In Canada, call 1 800 465-7999.

Operate at a higher level.™



```
#ifndef ccall_idl
#define ccall_idl

#include "appt.idl"

interface ConferenceCall : Appointment
{
    attribute string phoneNumber;
    // The phoneNumber of the appointment.

    implementation {

        dllname = "apptbk.dll";
        releaseorder: _get_phoneNumber, _set_phoneNumber;

    // Method Modifiers
        somInit: override;
        somUninit: override;
        bufferize: override;

    // Attribute Modifiers
        phoneNumber: noset;

    // Instance data
        string bp; // buffer used by bufferize method
    };
};

#endif /* ccall_idl */
```

Figure 14. Conference call definition: *ccall.idl*

```
#define ConferenceCall_Class_Source
#include <ccall.ih>

#define BUF_SIZE 512

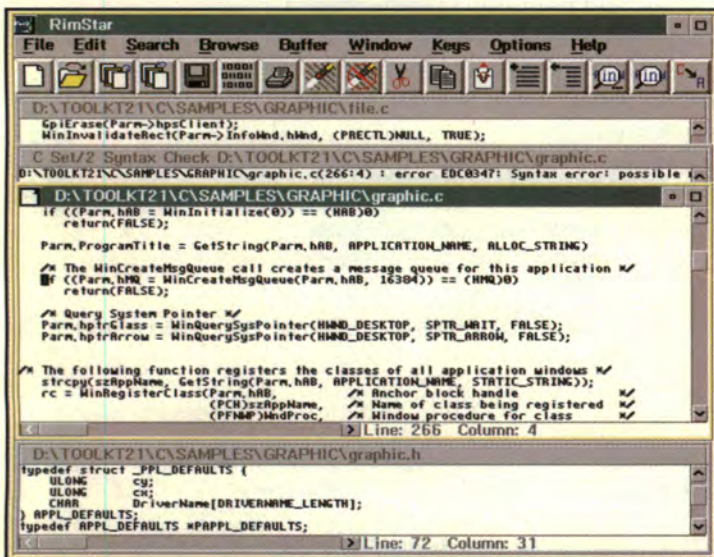
SOM_Scope void SOMLINK _set_phoneNumber(ConferenceCall somSelf,
                                         Environment *ev, string phoneNumber)
{
    ConferenceCallData *somThis = ConferenceCallGetData(somSelf);
    ConferenceCallMethodDebug("ConferenceCall", "_set_phoneNumber");

    if(_phoneNumber != NULL){
        SOMFree(_phoneNumber);
        _phoneNumber = NULL;
    }
    if(phoneNumber == NULL) return;
    _phoneNumber = SOMMalloc(strlen(phoneNumber)+1);
    strcpy(_phoneNumber, phoneNumber);
}
```

Figure 15. Conference call implementation (continued on page 24)

Brief's performance is over. Meet the new star.

The RimStar™ Programmer's Editor V2.0



If you're looking for a fully configurable, professional OS/2 PM programmer's editor to replace your current text-mode editor – we have what you have been looking for!

No hassle changing editors

We know changing editors can be a major hassle. You don't want to relearn a new set of keystrokes no matter how good the product. Well, you don't have to – we have Brief, Epsilon, Multi-Edit, SlickEdit, PWB and Borland IDE keyboard mapping waiting for you. If you're not using one of these, let us know and we will create the mappings you need.

Features designed to increase productivity

It has all the features you have come to expect, plus special features designed to anticipate your needs and increase your productivity. Features like a 'C' source browser that eliminates those time consuming searches for functions, global variables, typedefs, and macros by providing you with instant positioning to both definitions and references. Using our powerful 'C' macro language you can customize your programming environment to suit your individual needs.

All the features you need

- ✓ Complete 'C' macro language
- ✓ Auto-indent & template editing
- ✓ No limits on files, file size, or windows
- ✓ Bookmarks
- ✓ Line lengths to 16K
- ✓ Timed auto save
- ✓ Access PDK help for function under cursor
- ✓ Multi-threaded for no waiting
- ✓ Configurable tool bar
- ✓ Import/export to system clipboard
- ✓ Keystroke record/playback
- ✓ Block indent/outdent
- ✓ Brace matching
- ✓ Customizable menus
- ✓ Column, line and block selection, search and replace
- ✓ Integrates with Workframe/2
- ✓ Source browser for 'C'
- ✓ Unlimited undo and redo
- ✓ Multi-buffer regular expression search and replace
- ✓ Compile and jump to errors
- ✓ Complete on-line help
- ✓ Save and restore states between sessions
- ✓ OS/2 2.x 32 bit PM Multi-document interface

"My copy of Brief has been permanently retired. Keep up the good work!" - A.L.

Upgrade from your current editor now for only \$99.00*

*Offer expires 12/31/93. Regular list price \$299.

RimStar Technology, Inc.
91 Halls Mill Road
Newfields, NH 03856-0938

To order call (603) 778-2500.
90 day money-back guarantee.

All products and company names are trademarks or registered trademarks of their respective holders.

© 1993 RimStar Technology Inc.



add a `phoneNumber` attribute instead of a `location` attribute.

Figure 15 shows the `ConferenceCall` implementation. This closely parallels the implementation of `Meeting`.

Next we look at the definition of `AppointmentBook`. As we discussed, `AppointmentBook` can take a number of appointments, any one of which can be either an `Appointment`, a `Meeting`, or a `ConferenceCall`. Actually, our system is set using `Appointment` as a kind of general base class, and we assume actual instantiations will be either `Meeting` or `ConferenceCall`.

The definition of `AppointmentBook` is shown in Figure 16. There are two new syntactic constructions we have not seen before: `const` and `sequence`. The `const` construction allows us to define a constant value. The `sequence` allows us to define a simple collection of items.

The `sequence` is basically a C structure containing three elements. One of the elements is a pointer to a buffer containing an array. Unless the `sequence` is of a simple type (such as short or long), the `nth` item of the array is a pointer to the `nth` thing in the `sequence`. The second element in the structure is the number of items in the array and the third is the size of the array. The size of the array minus the number of elements already in the array tells us the number of empty slots in the `sequence`.

SOM automatically defines a set of macros that make it easier to manage sequences. They are:

- `sequenceLength(sequence)`, which returns the length of a `sequence`
- `sequenceMaximum(sequence)`, which returns the size of the array,
- `sequenceElement(sequence,n)`, which returns the `nth` element of the array.

Since you will do most of your `sequence` access using these macros, the only `sequence` element you really need to know about is `sequence.buffer`, a pointer to the array containing the pointers to the `sequence` items.

As you can see from Figure 16, `AppointmentBook` defines four methods.

```
}

SOM_Scope void SOMLINK somInit(ConferenceCall somSelf)
{
    ConferenceCallData *somThis = ConferenceCallGetData(somSelf);
    ConferenceCallMethodDebug("ConferenceCall","somInit");

    ConferenceCall_parent_Appointment_somInit(somSelf);
    _phoneNumber = NULL;
    _bp = SOMMalloc(BUF_SIZE);
}

SOM_Scope void SOMLINK somUninit(ConferenceCall somSelf)
{
    ConferenceCallData *somThis = ConferenceCallGetData(somSelf);
    ConferenceCallMethodDebug("ConferenceCall","somUninit");

    if(_phoneNumber != NULL){
        SOMFree(_phoneNumber);
        _phoneNumber = NULL;
    }
    ConferenceCall_parent_Appointment_somUninit(somSelf);
}

SOM_Scope string SOMLINK bufferize(ConferenceCall somSelf, Environment
*ev)
{
    string pbp;
    ConferenceCallData *somThis = ConferenceCallGetData(somSelf);
    ConferenceCallMethodDebug("ConferenceCall","bufferize");

    pbp = ConferenceCall_parent_Appointment_bufferize(somSelf, ev);
    _bp = SOMRealloc(_bp, BUF_SIZE + strlen(pbp));
    sprintf(_bp, "%sPhone Number: %s\n", pbp, _phoneNumber);
    return(_bp);
}
```

Figure 15. Conference call implementation (continued from page 22)

`addAppointment()` takes an `Appointment` and adds it to the `AppointmentBook`. `getDaysAppointments()` takes information about a particular date and returns a `sequence` of all the `Appointments` that occur on that date. Note that the `sequence` this method returns is a subset of the `sequence` attribute the `AppointmentBook` uses to internally manage its `Appointments`.

The last method is `deleteAppointment()`. You can delete an appointment by invoking `deleteAppointment()`, passing

in the ID of the appointment to be deleted. The `Appointment` ID is actually a short integer that is one of the data elements in `Appointment`. We have not talked about it much since its only purpose is to support the `deleteAppointment()` method.

In our implementation, shown in Figure 17, `AppointmentBook` is responsible for assigning IDs to `Appointments`. It does this by starting with zero and incrementing the ID of each `Appointment` as it adds it. Notice that the `AppointmentBook`

THE PARALLEL SOLUTION



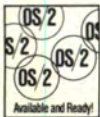
TAPE BACKUP

plugs into any standard parallel port, supports DOS, OS/2, Windows, NetWare, SCO UNIX and XENIX.



• Complete with powerful solution-minded software series, including BUSS (BackUp Supervisor Software), SM (Script Manager) and SDB (SCSI Disk Backup).

• Timesaving, simultaneous backup and verify at true 11MB/min.



• 250MB-4GB capacities in native mode.

• Optical 128MB-1.2GB, removable media.

• QIC industry standard format.

• Extremely portable, weighs as little as a notebook.



PSS PARALLEL STORAGE SOLUTIONS

The Pacesetter in Parallel to SCSI Technology

INFO AND ORDERS: 1-800-998-7839

CORPORATE HEADQUARTERS • 116 S. CENTRAL AVE • ELMSFORD, NY USA 10523 • FAX: (914) 347-4646

OS2TREE™

See your entire system graphically at a glance!

"What XTREE GOLD is to DOS and More!

- Completely programmable: every function/key can be customized!
- Display your tree structure for all system and LAN disks concurrently!
- Manipulate your files, directories, entire disks and/or entire system!
- HPFS fully supported
- Pop-up list of files for any directory
- Edit/browse file(s) using any EDITOR or PROGRAM!
- Extensive FILE Search and Tree directory search capabilities make locating files and/or data easy.
- Upload/download files between mainframe and PC
- Secure delete (sensitive files wiped clean)

Intro Price: \$79.99

Professional Version: \$249.99 Limited time special \$180.00

Network Licenses Available

LEVINE COMPUTER CONSULTING SERVICES

7640 Provincial Dr., Suite 213, McLean, VA 22102

Orders only (800) 383-9495

FAX/Inquiries/HelpLine (703) 790-1660

SQL Objects++ Database Class Library

Inherit The Power™

SQL Objects++ Database Class Library is a powerful collection of object-oriented database classes that provide *full* access to SQL and non-SQL databases:

- Oracle
- Sybase
- SQL Server
- Netware SQL
- Btrieve
- DDCS/2 (DB2, SQL/400, SQL/DA)
- Others

The Object-Oriented

Approach. No longer are you restricted to the lowest common denominator factor when developing database applications. Inherit the power *now*. Derive new classes with powerful custom extensions. Realize more flexibility than ever before. Use Object Technology today.

Multi-Platform Support.

Whether you use OS/2, DOS, Windows or NT, SQL Objects++ Database Class Library is designed to take advantage of your native environment.

NO RISK and NO ROYALTIES.

Use it for 30 days. If your not satisfied, return it for a full refund. Order today for only \$499.

**SPECIAL
\$249**

Limited Time Offer

GSI Graphical
Software
Interfaces, Inc.

47 Stonewall Street Cartersville, GA 30120

Sales: (800) 876-6585

(404) 382-6585 fax (404) 382-6374

SQL Objects++ and Inherit The Power are trademarks of Graphical Software Interfaces, Inc. All company and product names are trademarks or registered trademarks of their respected owners.

10 Reasons to Become a CSI Member

BECOME A CSI MEMBER AND TAKE ADVANTAGE OF ALL THESE BENEFITS

1. The Computer Security Alert—

CSI's monthly members-only newsletter that provides timely news and practical insights you can put to use immediately.

2. The Computer Security Journal—

The semi-annual journal of comprehensive, practical articles, case studies, reviews and commentaries, written by knowledgeable computer security pros covering the full spectrum of information security topics.

3. Educational Discounts—

Members receive special discounts and benefits on CSI conferences, one- and two-day seminars and training.

4. Members' Hotline—For security emergencies, problems or questions, the CSI Members' Hotline is your direct connection to practical advice, resources and referrals.

5. Networking Opportunities—

Meet experienced colleagues who have already encountered and solved some of today's toughest security problems. Many CSI members report that networking is the #1 benefit of their membership.

6. Career Enhancement—As a CSI member, you have the most up-to-date information on computer security at your fingertips. CSI is a solid support resource you can count on—giving you expert, practical advice and helping you build your individual successes.

7. The Annual Computer Security

Buyers Guide—The most comprehensive resource available on computer security products and services.

8. On-line Bulletin Board—

Our BBS lets you access the latest computer security news, product announcements, industry alerts, technical updates and job listings.

9. The Manager's Guide Series—

Five copies each of these timely, non-technical explanations of computer security topics that include VirusThreats, Security Awareness, Telecommunications Fraud, Privacy and Encryption.

10. Publications Discounts—

Receive savings on books including The Computer Security Handbook & many other publications.



C O M P U T E R
S E C U R I T Y
I N S T I T U T E

600 Harrison Street
San Francisco, CA 94107

Phone: (415) 905-2626
FAX: (415) 905-2218


```

#ifndef apptbook_idl
#define apptbook_idl

#include <appt.idl>

interface AppointmentBook : SOMObject
{
    const short APPOINTMENTS_INCR = 16;

    attribute sequence<Appointment> appointments;

    short addAppointment(in Appointment newAppointment);
    // Add an appointment to the book.
    // Returns an appointment number.

    sequence<Appointment> getDaysAppointments(in short year,
                                                in short month, in short day);
    // Return a sequence of the given day's appointments.

    void deleteAppointment(in short appointmentNumber);

```

Figure 16. AppointmentBook definition: apptbook.idl (continued on page 28)

also contains a short attribute named `apptId` that keeps track of the last assigned Appointment ID.

The first method in the implementation file is `addAppointment()`. It is passed an `Appointment`, updates the ID of the `Appointment`, and then adds the `Appointment` to the appointments sequence. The only difficult part of adding the `Appointment` to the sequence is dealing with a situation in which the sequence is already filled. In this case, we reallocate the buffer size to be large enough for an additional `APPOINTMENTS_INCR` number of appointments and reset the `sequenceMaximum`.

The next method in the implementation is `getDaysAppointments()`, which starts by allocating a sequence large enough to hold all the appointments in a given day. It then checks each element of the appointments sequence and, if the date of



So many
industry leaders
agree on
one object database,



Scott McNealy
Chairman and CEO
Sun Microsystems

Ray Noorda
President and CEO
Novell, Inc.

Steve Mills
General Manager
IBM Software Solutions

isn't it worth
a phone call?
1-800-962-9620

It sure is. Because when you call Object Design® you'll get the inside story on the object database from a company Ray Noorda called "the driving force behind the future of ODBMS," Scott McNealy described as "the foundation for a new generation of enterprise-wide applications" and Steve Mills declared is "the leader in object database technology."

It's called ObjectStore™, and discovering what these leaders already know is easy. Just call for your free Object Database Evaluation Kit, featuring an Evaluation Guide, user stories and our video.

With the Kit, you'll learn what to look for in an object database. You'll see the importance of interactive performance, scalability and persistence independent of type. And you'll discover how our innovative virtual memory mapping architecture delivers those benefits, making ObjectStore the world's leading object database.

It's all in the Kit. Along with details on exclusive features. So if you're ready to choose an object database, call for your Evaluation Kit now. And see what everyone is talking – and agreeing – about.



Get your free Kit now!

**1-800
962-9620**
Or e-mail
info@odi.com

Object Design, Inc.
Corporate Headquarters
Burlington, MA USA

Wiesbaden, Germany: 49-611-39707-0
Swindon, UK: 44-793-486111
Sydney, Australia: 61-2-212-2766
Tokyo, Japan: 81-3-3251-2882

© 1993 Object Design, Inc. Object Design and Leadership by Design are registered trademarks and ObjectStore is a trademark of Object Design, Inc.


OBJECT DESIGN
Leadership by Design®


```

// Delete the given appointment from the book.

attribute short apptId;
// Appointment book id generator.

implementation {

    dllname = "apptbk.dll";
    releaseorder: addAppointment, getDaysAppointments,
                  deleteAppointment,
                  _get_appointments, _set_appointments,
                  _get_apptId, _set_apptId;

// Method Modifiers
    somInit: override;
    somUninit: override;

// Instance data
    sequence<Appointment> daysAppointments;
};
};

#endif /* apptbook_id1 */

```

Figure 16. AppointmentBook definition: *apptbook.id1* (continued from page 27)

```

#define AppointmentBook_Class_Source
#include <appt.h>
#include <apptbook.ih>

SOM_Scope short SOMLINK addAppointment(
    AppointmentBook somSelf, Environment *ev, Appointment newAppointment)
{
    AppointmentBookData *somThis = AppointmentBookGetData(somSelf);
    AppointmentBookMethodDebug("AppointmentBook", "addAppointment");

/* Generate an id for the newAppointment.
----- */
    _apptId++;
    Appointment__set_apptId(newAppointment, ev, _apptId);

/* Insert newAppointment into the book.
----- */
    if (sequenceLength(_appointments) == sequenceMaximum(_appointments)) {
        sequenceMaximum(_appointments) += APPOINTMENTS_INCR;
        _appointments._buffer = (Appointment *)
            SOMRealloc(_appointments._buffer,
                sizeof(Appointment) * sequenceMaximum(_appointments));
    }
    sequenceElement(_appointments, sequenceLength(_appointments)) =
        newAppointment;
    sequenceLength(_appointments)++;

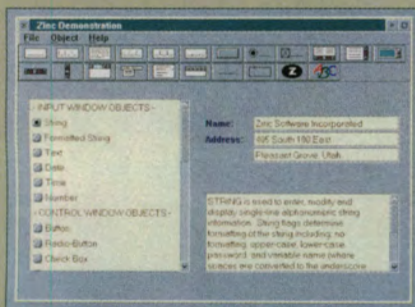
    return(_apptId);
}

```

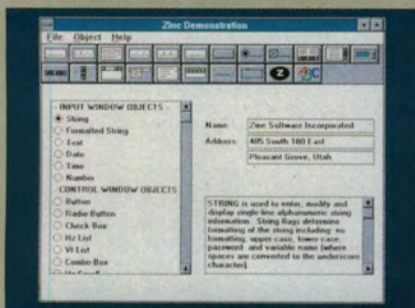
Figure 17. AppointmentBook implementation (continued on page 30)


 THINK
ZINC

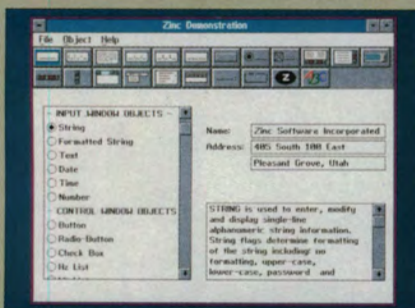

UNIX MOTIF



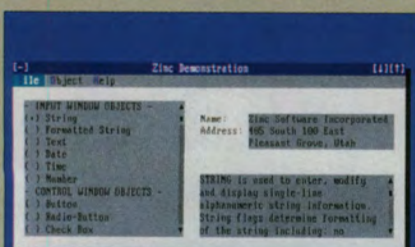
OS/2 2.0



MS WINDOWS & WINDOWS NT



DOS GRAPHICS



DOS TEXT

ZINC™ APPLICATION FRAMEWORK™ 3.5

Multiplatform Flexibility From One Application Framework.

FLEXIBILITY is an essential component in software design. Your development tools should allow you to easily incorporate new features and technologies into your applications without rewriting all of your code. That's a tall order if your development tools are designed like a straight-jacket. Zinc Application Framework 3.5 and Zinc Designer™ give you flexible and extendible support for Microsoft Windows, Windows NT, OS/2 2.0, UNIX Motif, DOS Graphics and DOS Text. And Zinc does it with ONE set of source code. Zinc's multiplatform, object-oriented architecture won't confine your application development options today... or tomorrow.

FOR A FREE ZINC DEMO KIT, CALL US TOLL FREE TODAY AT

1.800.638.8665. IN EUROPE CALL +44 (0) 81 855 9918

z i n c

ZINC SOFTWARE INCORPORATED, 405 SOUTH 100 EAST, 2ND FLOOR, PLEASANT GROVE, UTAH 84062, TEL 801.785.8900, FAX 801.785.8996, BBS 801.785.8997.

EUROPE: ZINC SOFTWARE (UK) LIMITED 58-60 BERESFORD STREET, LONDON SE18 6BG, TEL +44 (0) 81 855 9918, FAX +44 (0) 81 316 7778, BBS +44 (0) 81 317 2310

© COPYRIGHT 1992 ZINC SOFTWARE INCORPORATED, ALL RIGHTS RESERVED. ZINC, ZINC DESIGNER AND ZINC APPLICATION FRAMEWORK ARE TRADEMARKS OF ZINC SOFTWARE INCORPORATED. OTHER TRADEMARKS ARE OWNED BY THEIR RESPECTIVE COMPANIES.


```

SOM_Scope_IDL_SEQUENCE_Appointment SOMLINK getDaysAppointments(
    AppointmentBook somSelf, Environment *ev,
    short year, short month, short day)
{
    AppointmentBookData *somThis = AppointmentBookGetData(somSelf);
    int i;
    Appointment appt;
    AppointmentBookMethodDebug("AppointmentBook","getDaysAppointments");

    /* Allocate the returned buffer.
    ----- */
    _daysAppointments._buffer = (Appointment *)
        SOMRealloc(_daysAppointments._buffer,
            sizeof(Appointment) * sequenceLength(_appointments));
    sequenceMaximum(_daysAppointments) = sequenceLength(_appointments);
    sequenceLength(_daysAppointments) = 0;

    /* Fill in the sequence with the days appointments.
    ----- */
    for (i= 0; i< sequenceLength(_appointments); i++) {
        appt = sequenceElement(_appointments, i);
        if ((__get_year(appt, ev) == year) &&
            (__get_month(appt, ev) == month) &&
            (__get_day(appt, ev) == day)) {
            sequenceElement(_daysAppointments, sequenceLength(_daysAppointments)) =
                appt;
            sequenceLength(_daysAppointments)++;
        }
    }
    return(_daysAppointments);
}

SOM_Scope void SOMLINK deleteAppointment(
    AppointmentBook somSelf, Environment *ev, short appointmentNumber)
{
    AppointmentBookData *somThis = AppointmentBookGetData(somSelf);
    short i, j;
    Appointment appt;
    AppointmentBookMethodDebug("AppointmentBook","deleteAppointment");

    for (i=0; i< _appointments._length; i++) {
        appt = sequenceElement(_appointments, i);
        if (Appointment__get_apptId(appt, ev) == appointmentNumber) {
            -- sequenceLength(_appointments);
            SOMFree(_appointments._bufferUie);
            for (j=i; j<sequenceLength(_appointments); j++) {
                sequenceElement(_appointments, j) =
                    sequenceElement(_appointments, j+1);
            }
            return;
        }
    }
}

SOM_Scope void SOMLINK somInit(AppointmentBook somSelf)
{
    AppointmentBookData *somThis = AppointmentBookGetData(somSelf);
    AppointmentBookMethodDebug("AppointmentBook","somInit");
}

```

Figure 17. AppointmentBook implementation (continued on page 31)


```

AppointmentBook_parent_SOMObject_somInit(somSelf);

_apptId = 0;
_appointments._buffer = (Appointment *)
    SOMMalloc(sizeof(Appointment) * APPOINTMENTS_INCR);
sequenceMaximum(_appointments) = APPOINTMENTS_INCR;
sequenceLength(_appointments) = 0;
_daysAppointments._buffer = (Appointment *)
    SOMMalloc(sizeof(Appointment) * APPOINTMENTS_INCR);
}

SOM_Scope void SOMLINK somUninit(AppointmentBook somSelf)
{
    AppointmentBookData *somThis = AppointmentBookGetData(somSelf);
    int i;
    AppointmentBookMethodDebug("AppointmentBook", "somUninit");

    for (i=0; i<sequenceLength(_appointments); i++) {
        SOMFree(sequenceElement(_appointments, i));
    }
    SOMFree(_appointments._buffer);
    SOMFree(_daysAppointments._buffer);

    AppointmentBook_parent_SOMObject_somUninit(somSelf);
}

```

Figure 17. AppointmentBook implementation (continued from page 30)

the Appointment matches the requested date, copies it into the sequence it will return.

The next method is deleteAppointment(). We delete by appointment ID. The method looks through the appointments, searching for one whose ID matches the parameter. When found, it is deleted from the sequence and the other elements of the sequence are pushed together to remove blank space.

The next method is somInit(), an override of one of the methods inherited from SOMObject. This method first invokes the somInit() of its base class and then initializes its own instance data. _apptId is initially set to zero; newly added appointments, therefore, start with ID zero. Following this, the array portion of the appointment sequence maintained by this class are allocated to an initial size of APPOINTMENTS_INCR elements.

The next method is somUninit(), another override of one of the SOMObject methods. This goes through appointments

```

#include <som.h>
#include <appt.h>
#include <mtg.h>
#include <ccall.h>
#include <apptbook.h>

#ifdef __IBMC__
    #pragma linkage
        (SOMInitModule, system)
#endif

SOMEXTERN void SOMLINK SOMInitModule
(long majorVersion,
 long minorVersion)
{
    AppointmentNewClass(0,0);
    MeetingNewClass(0,0);
    ConferenceCallNewClass(0,0);
    AppointmentBookNewClass(0,0);
}

```

Figure 18. initmod.c

Looking for fast answers to your development questions?

**You need to do two things:
Go on-line with CompuServe
Information Service. Use Golden
CommPass to do it.**

CompuServe hosts OS/2 developer and user forums monitored by technical personnel. IBM developers are there with responses to anything that's got you stuck. Other OS/2 programmers and enthusiasts are there, too, comparing notes and giving feedback. It's definitely the place to be.

Time is money when you connect to CompuServe, and Golden CommPass saves you both. It lets you compose your questions off-line, send them in a flash and disconnect. It gets your replies while you're busy programming. It's a native OS/2 app, with all the power and features you'd expect.

Get the fastest access to answers. Golden CommPass keeps your development schedule on course. The fact is, the sooner you start using our program, the sooner they'll be talking about yours!

Phone:
(609) 234-1500
Fax:
(609) 234-1920



CompuServe:
71511, 151
90 Day Satisfaction
Guarantee

Creative Systems Programming Corporation


```

#include <appt.h>
#include <apptbook.h>
#include <mtg.h>
#include <ccall.h>

#include <stdio.h>
#define MAX_LENGTH 80
#define MAX_BUFFER_SIZE 1000

/* Prints a message and reads a line into buffer.
   ----- */
static void getLine(buffer, msg)
char buffer[];
char *msg;
{
    int i,c;
    somPrintf(msg);
    for (i=0; i<(MAX_LENGTH-1) && (c=getchar())
        != EOF && c!= '\n'; ++i)
        buffer[i] = c;
    buffer[i] = '\0';
}

void setUpAppointment(Appointment appt)
{
    char input[MAX_LENGTH];
    Environment *ev = somGetGlobalEnvironment();

    getLine(input, "Enter month > ");
    __set_month(appt, ev, atoi(input));

    getLine(input, "Enter day > ");
    __set_day(appt, ev, atoi(input));

    getLine(input, "Enter year > ");
    __set_year(appt, ev, atoi(input));

    getLine(input, "Enter starting time > ");
    __set_start(appt, ev, atoi(input));

    getLine(input, "Enter ending time > ");
    __set_end(appt, ev, atoi(input));

    getLine(input, "Enter subject > ");
    __set_subject(appt, ev, input);
}

main ()
{
    short id;
    short i;
    short day, month, year;
    Appointment appt;
    AppointmentBook apptBook = AppointmentBookNew();
    _IDL_SEQUENCE_Appointment apptList;
    char menu[MAX_LENGTH];
    char input[MAX_LENGTH];
    string buffer;
    string menuLine =
        "\nm(meeting) c(conference) r(remove) d(display) q(quit) > ";
    Environment *ev = somGetGlobalEnvironment();
    do{

/* Display the menu options and read the operation.
   ----- */

```

Figure 19. Client program (continued at right)

```

getLine(menu, menuLine);
switch(menu[0]){

/* Set up a meeting.
   ----- */
case 'm':
    appt = MeetingNew();
    setUpAppointment(appt);
    getLine(input, "Enter location > ");
    __set_location(appt, ev, input);
    id = _addAppointment(apptBook, ev, appt);
    somPrintf(
        "Appointment id for this meeting is : %d \n", id);
    break;

/* Set up a conference call.
   ----- */
case 'c':
    appt = ConferenceCallNew();
    setUpAppointment(appt);
    getLine(input, "Enter phone number > ");
    __set_phoneNumber(appt, ev, input);
    id = _addAppointment(apptBook, ev, appt);
    somPrintf(
        "Appointment id for this conference call is : %d
\n",
        id);
    break;

/* Remove an appointment.
   ----- */
case 'r':
    getLine(input, "Enter appointment id > ");
    id = atoi(input);
    _deleteAppointment(apptBook, ev, id);
    break;

/* Display the appointments for a given day.
   ----- */
case 'd':
    getLine(input, "Enter appointment month > ");
    month = atoi(input);
    getLine(input, "Enter appointment day > ");
    day = atoi(input);
    getLine(input, "Enter appointment year > ");
    year = atoi(input);
    apptList = _getDaysAppointments(apptBook, ev,
        year, month, day);
    for (i=0; i< sequenceLength(apptList); i++) {
        buffer = _bufferize(sequenceElement(apptList,i), ev);
        somPrintf("\n%s", buffer);
    }
    break;

/* Clean up and exit.
   ----- */
case 'q':
    break;

}
} while (menu[0] != 'q');
_somFree(apptBook);
}

```

Figure 19. Client program (continued from left)

sequence freeing each element and, finally, the buffer itself. It also frees the sequence buffer used to return a day's worth of appointments.

The last piece of code needed to implement a class library is the `SOMInitModule` procedure, one of which must be implemented for each class DLL. `SOMInitModule` is called by SOM after it dynamically loads your DLL. This function is expected to initialize your class library. The `SOMInitModule` procedure must call the `classname NewClass` procedure of each of the classes implemented in your DLL. Our `SOMInitModule`, shown in Figure 18, calls the `NewClass` procedure of each of our appointment book classes.

Figure 19 shows a client program working with an appointment book. The main program is a loop following the algorithm:

```
set up appointment book;

do { show them what they can do;
    ask what they want to do;
    if (they want to set up meeting)
        create meeting appointment;
    if (they want to set up conference call)
        create conference call;
    If (they want to delete an appointment)
        delete appointment;
    If (they want to display a days appointments)
        show days appointments;
} while (they are not done)
```

The most interesting part of the code is the display of the day's appointments. We go through the sequence asking each element to bufferize itself, then print out the resulting buffer. Notice how this depends on a polymorphic resolution of the method `bufferize()`.

The `make` file that creates both the client program (named `SAMPLE.EXE`) and the class library (named `APPTBK.DLL`) is shown in Figure 20. The `make` file is designed to be easily adapted to other SOM projects.

Figure 21 shows the module

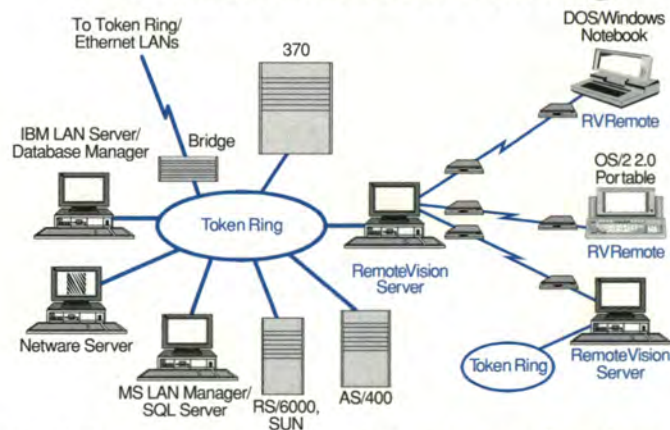
definition file used to create the `APPTBK.DLL`.

Figure 22 shows the output from one run of the program. We add a meeting and a conference call, then dis-

play all appointments for 9/17/1993. Notice the polymorphic resolution of `bufferize()`, indicated by the fact that the first displays the location and the second the phone number.



Introducing RemoteVision.[™] The Complete Software Solution For Remote Networking.



- Extend LANs transparently over asynchronous modems to remote workstations and workgroups.
- Run DOS, Windows[™], & OS/2[™] apps written to multiple protocol stacks on remote nodes.
- Operate remote computers as full-function LAN nodes.
- Connect to remote offices with LAN-to-LAN dial-up networking.
- Add remote connectivity services using existing LAN machines.
- Access remote PC servers and hosts without additional host gateways.
- No specialized hardware/adapters.

RemoteVision makes it easy and affordable for DOS, Windows and OS/2 users in remote locations to connect to Token Ring or other LANs using dial-up modems. Response time remains fast, applications don't need changing, and users won't need retraining. So you can turn your remote machines into full-function workstations. With RemoteVision. To immediately find out more, or to take advantage of our "Try Now, Pay Later" 30-day money-back guaranteed trial offer, call today.



1265 Montecito Ave., Ste. 101, Mountain View, CA 94043
Tel: (415) 965-8607, Fax info: (415) 965-8607, (then press 2)
Trademarks are from their respective companies. ©1993 Token Technology, Inc. 001-0


```

OBJS      = sample.obj
TARGET    = sample.exe
DLLOBJS   = appt.obj mtg.obj ccall.obj apptbook.obj initmod.obj
DLL        = apptbk
CFLAGS    = -Q+ -W3 -D_MIG_LIB__ -Ti+
LDFLAGS    = /packd /packc /exepack /align:16 /noi /m $(LDEB) /no1 \
             /PM:VID /co
SCFLAGS    = -p -u -sh;ih;c
CC         = icc
LIBLIST    = $(SOMBASE)\lib\somtk os2386
CORBASTYLE = $(SOMBASE)\include\somcorba.bld

$(TARGET): $(CORBASTYLE) $(DLL).lib $(OBJS)
    link386 $(LDFLAGS) $(OBJS), $*, $*/m, $(LIBLIST) $(DLL).lib, nul

$(DLL).dll: $(CORBASTYLE) $(DLLOBJS) $(DLL).def
    link386 $(LDFLAGS) $(DLLOBJS), $0, $*/m, $(LIBLIST), $(DLL).def

$(DLL).lib: $(DLL).dll
    implib /nologo $0 $(DLL).def

.SUFFIXES: .c .h .idl .ih .obj .csc

.c.obj:
    $(CC) $(CFLAGS) -Ge- -c $<

.csc.ih:
    sc $(SCFLAGS) $*.csc

.idl.ih:
    sc $(SCFLAGS) $*.idl

.csc.h:
    sc $(SCFLAGS) $*.csc

.idl.h:
    sc $(SCFLAGS) $*.idl

clean:
    -del som.ir *.obj *.dll *.lib *.exe *.map *.ih $(CLEANFILES) >nul 2>&1

$(CORBASTYLE):
    @echo This program requires the CORBA-style C bindings
    @echo created using the somcorba command.
    @exit 1

appt.ih: appt.idl
appt.obj: appt.ih appt.c
mtg.ih: mtg.idl
mtg.obj: mtg.ih mtg.c
ccall.ih: ccall.idl
ccall.obj: ccall.ih ccall.c

apptbook.ih: apptbook.idl
apptbook.obj: apptbook.ih apptbook.c
initmod.obj: appt.h mtg.h ccall.h apptbook.h
sample.obj: sample.c appt.h mtg.h ccall.h apptbook.h

    $(CC) $(CFLAGS) -c $*.c

```

Figure 20. Makefile

```

LIBRARY apptbk INITINSTANCE

DESCRIPTION 'Appointment Book
            Class Library'

PROTMODE

DATA MULTIPLE NONSHARED
LOADONCALL

EXPORTS
    AppointmentCClassData
    AppointmentClassData
    AppointmentNewClass
    AppointmentBookCClassData
    AppointmentBookClassData
    AppointmentBookNewClass
    ConferenceCallCClassData
    ConferenceCallClassData
    ConferenceCallNewClass
    MeetingCClassData
    MeetingClassData
    MeetingNewClass

```

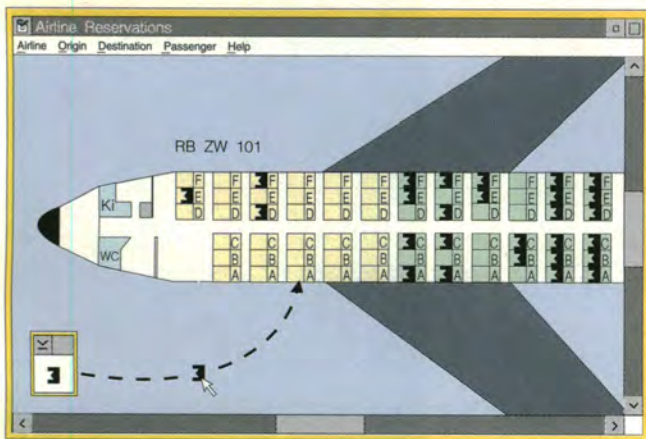
Figure 21. Module definition file

ACKNOWLEDGMENTS

SOM is the work of many people. Mike Conner developed the idea and implementation and continues to lead the overall design. Andy Martin designed and implemented the class interface language and compiler. Larry Raper implemented many features of the run-time library and ported SOM to OS/2. Larry Loucks provided close technical tracking and was instrumental in focusing team effort. The case study used here is based on three frameworks implemented with SOM, the persistence, replication, and record and playback frameworks. Other SOM developers include Scott Danforth, Hari Madduri, Liane Acker, and Kathy Bohrer, led by project managers Cliff Reeves, Rose Ann Roth, and Jerry Ronga.

This program contains scenes of a graphic nature.

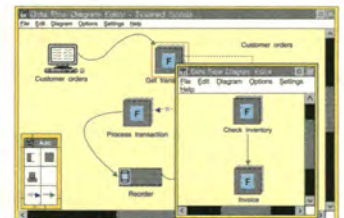
Introducing
Graphics
Interface
Kit/2



But don't worry. It's all in good clean fun. Because now there's a fast, cost-effective way to create graphic, easy-to-use interfaces for OS/2® 2.0 applications and tools. Introducing Graphics Interface Kit/2 (GIK/2), an AD/Cycle® product from IBM Programming Systems.

GIK/2 makes it easy to produce and manipulate symbolic representations of data without writing a single line of Presentation Manager® code. It automatically generates C code, where you can link the graphic symbols to the application data they represent. This not only speeds up your development cycle, it also helps you create applications that are intuitive and easy-to-use. Simplify transaction systems. Make databases easy to access. Bring organizational charts to life. Track inventory with simple icons. Any application can be made easier with the right graphic front-end. And GIK/2 can help you make it happen.

For orders or additional information, please call 1 800 IBM-CALL and ask for Department S71. If you need graphic proof, ask for our evaluation demo which, by the way, is rated G.





```
m(meeting) c(conference) r(remove) d(display) q(quit) > m
Enter month          > 09
Enter day            > 17
Enter year           > 1993
Enter starting time  > 1015
Enter ending time    > 1100
Enter subject        > Marketing Meeting
Enter location       > 7D004
Appointment id for this meeting is : 1

m(meeting) c(conference) r(remove) d(display) q(quit) > c
Enter month          > 09
Enter day            > 17
Enter year           > 1993
Enter starting time  > 0900
Enter ending time    > 1000
Enter subject        > Product Planning
Enter phone number   > 512 111 2222
Appointment id for this conference call is : 2

m(meeting) c(conference) r(remove) d(display) q(quit) > d
Enter appointment month > 09
Enter appointment day   > 17
Enter appointment year  > 1993

(1) Date: 9/17/1993 Start: 1015 End: 1100
Subject   : Marketing Meeting
Location  : 7D004
(2) Date: 9/17/1993 Start: 900 End: 1000
Subject   : Product Planning
Phone Number: 512 111 2222
m(meeting) c(conference) r(remove) d(display) q(quit) > q
```

Figure 22. Program output

Nurcan Coskun, IBM Corp., 11400 Burnet Rd., Austin, Texas 78758. Coskun's expertise is in integrated programming environments, code generators, incremental compilers, interpreters, language-based editors, symbolic debuggers, application frameworks, and language design. She has worked on the OS/2 Database Manager and is now working on object-oriented programming environments. She holds a B.S. in industrial engineering from Middle East Technical University and an M.S. and Ph.D. in computer science from the University of Missouri, Rolla. Coskun can be contacted on Internet at nurcan@ausvm1.vnet.ibm.com.

Roger Sessions, IBM Corp., 11400 Burnet Road, Austin, Texas 78758. Sessions is the author of *Reusable Data Structures for C*. His most recent book, *Class Construction in C and C++: Object-Oriented Programming Fundamentals*, was chosen as a main selection for the Library of Computer and Information Science book club. He has authored many articles and frequently lectures on object-oriented programming, C++, and SOM. He is working on persistent SOM frameworks and has previously worked with high-performance relational databases and object-oriented storage systems. Sessions has a B.A. from Bard College and an M.E.S. in database systems from the University of Pennsylvania. He can be contacted on Internet at roger@vnet.ibm.com.

REFERENCES AND BIBLIOGRAPHY

Coskun, Nurcan and Roger Sessions, "Class Objects in SOM", *OS/2 Developer* (Summer 1992): 67-77. Intermediate level.

Danforth, Scott, "A Bird's Eye View of SOM," *OS/2 Developer* (Spring 1993): 86-93. Advanced level.

Orfali, Bob and Dan Harkey, *Client/Server Programming with OS/2 2.0, Second Edition*. New York: Van Nostrand Reinhold, 1992. (IBM Doc. G325-0650) Intermediate level.

Sessions, Roger and Nurcan Coskun, "Object-Oriented Programming in OS/2 2.0," *IBM Personal Systems Developer* (Winter 1992): 107-119. Introductory level.

Sessions and Coskun, "SOM Enchanted Evening...", *Computer Language* (Supplement), April 1993, 7-15. Introductory level.

Sessions and Coskun, "Method Resolution in SOM," *OS/2 Developer* (Spring 1993), 94-102. Advanced level.

Sessions, Roger, *Class Construction in C and C++: Object-Oriented Programming Fundamentals*, Englewood Cliffs, N.J.: Prentice-Hall, 1992. (IBM Doc. S242-0086, ISBN 0-13-630104-5) Good introduction to object-oriented programming for C programmers.

Charles Erickson, IBM Corp., 11400 Burnet Rd., Austin, Texas 78758. Erickson is a staff programmer who is currently developing SOM frameworks in IBM's Object Technology Products group in Austin. He joined IBM in 1984 and helped develop the 5250 emulators in PC Support and the OS/2 Communications Manager. He received a B.S. in computer science from Iowa State University.

GET ON A CLIENT/SERVER TRACK WITHOUT DERAILING YOUR DEVELOPMENT

If you're client/server bound, KASE:VIP visual design and code generation tools get your applications on an OS/2 or Windows GUI-based track — while leveraging your current development assets, your existing staff and even your existing application logic.

KASE:VIP automatically generates all the source code you need to build seamless links between graphical interfaces and SQL databases or CICS transactions. In standard languages like C, C++ and COBOL, so you can avoid the derailing overhead and restrictions of proprietary languages, runtimes and royalties.

KASE:VIP also offers an OPEN ARCHITECTURE license that gives you the flexibility to control the code generation process and accelerate your development. You can prototype and automatically generate code unique to your needs — such as specific line-of-business functions, proprietary APIs or your own application "hooks." All of which lets you leverage the expertise of key developers across your entire organization.

Avoid development derailment. Use KASE:VIP to keep your client/server development on the fast track.

Get your GUI-based SQL and CICS client/server development on a COBOL, C or C++ fast track — without proprietary languages or runtimes. Call **1-800-888-4335** for a KASE:VIP demonstration disk, or for information about KASEWORKS seminars in your area.

KASEWORKS™

△△△△ Enabling Client/Server Strategies

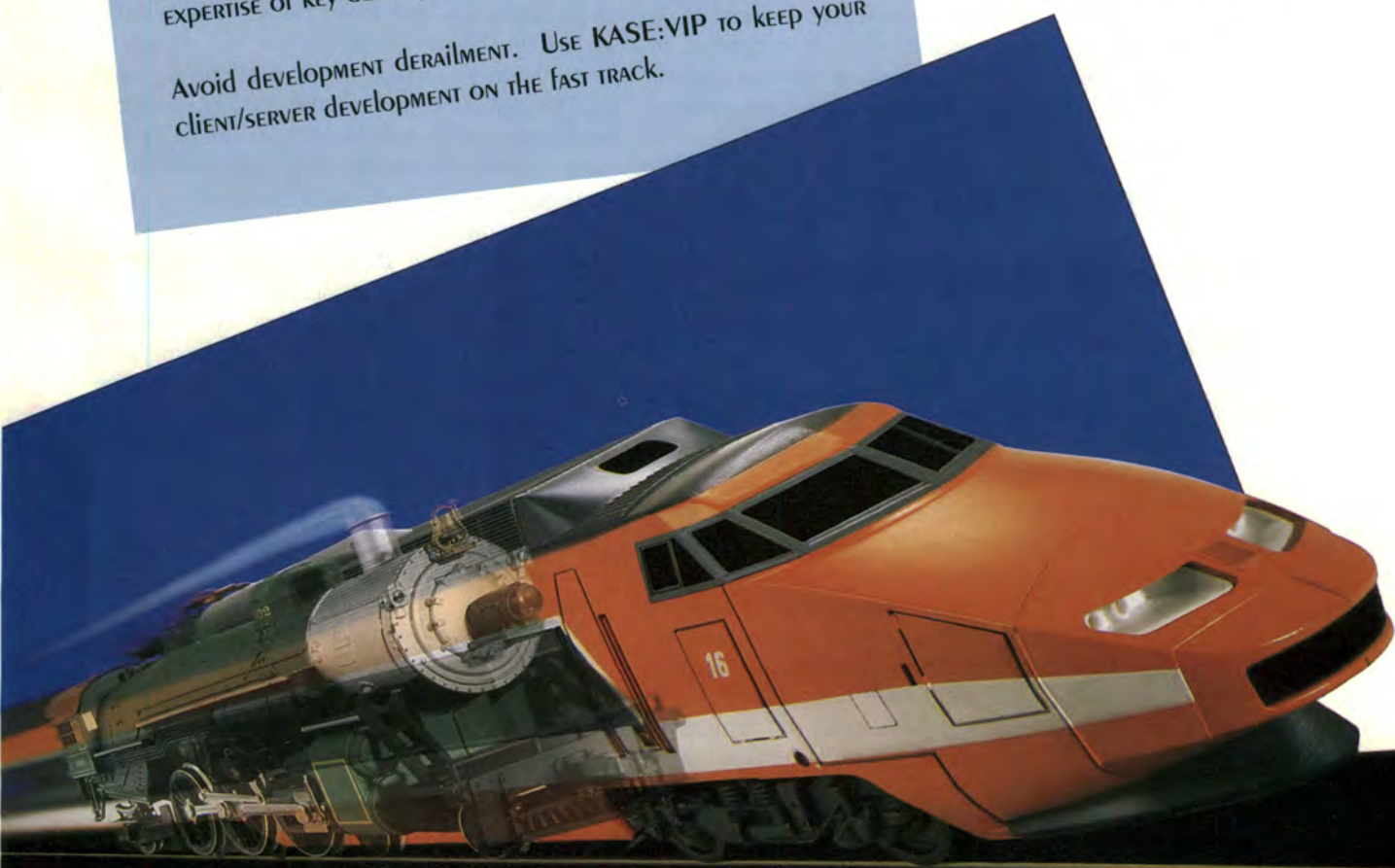
(FORMERLY CASEWORKS)

3295 RIVER EXCHANGE DRIVE, SUITE 430

NORCROSS, GA 30092

(800) 888-4335

(404) 448-4240





Object-Oriented Programming

A new generation of object-oriented database management systems, or ODBMS, meets the demands of compute-intensive applications such as engineering design (CAD, CASE, and so on), telecommunications, and complex document management. This article gives an overview of ODBMS and illustrates how existing applications can be migrated from OS/2 file-based storage to an ODBMS. **BY MOHAMMED ANWARUDDIN**

Migrating From File-Based to Object Database Storage On OS/2

Object-oriented database management systems are becoming widely used with many applications and operating environments. In fact, it is almost impossible to qualify the many types of applications for which object databases could have value. Early users have come from several key fields (primarily engineering design, telecommunications, and document management) but are now found in areas such as financial products and process control reporting. In general, an object database may make sense for your application if some or all of the following criteria are met:

Object-oriented databases integrate object-oriented language facilities with database capabilities such as persistence, concurrency, and integrity.

1. Traditional DBMS technology has not been appropriate because of performance demands or programming complexity.
2. The amount of information is large or complex enough that DBMS capabilities such as recovery/restart, transaction consistency, or nonstop operations are required.
3. The application requires multiuser concurrent access to data at a finer granularity than that of

an operating system file.

The level of complexity of data-intensive applications such as CAD, CASE, and Network Management—and the data they manipulate—has grown beyond the capabilities of traditional database systems, which are record-oriented and limited by a finite set of data types.

Object-oriented languages are ideal for these applications because they offer extensible data-typing capability, which facilitates the storage of information not suitable for conventional databases. Through the object-oriented concept of inheritance, behavior and representation can be passed from one object to another. This allows code sharing among software modules and structure sharing among data objects.

Object-oriented databases integrate these object-oriented language facilities with the database capabilities of persistence, concurrency, transactions, recovery, querying, versioning, and integrity. There are a number of approaches for integrating object-oriented capabilities into databases. One obvious approach is to extend an existing object-oriented language with database capabilities. Integrating C++, for example, with these capabilities has several advantages:

- C++ users can use the ODBMS with very little extra effort.
- With a single type system, you can utilize the compile-time type-checking of C++ to provide type-checking for persistent as well as transient data.
- You can use the full power of C++, including



virtual functions, inheritance, polymorphism, encapsulation, and parameterized types for application development.

- It promotes reusability of code because the same code operates on either persistent or transient data. More important, libraries developed for manipulating transient data can be used with persistent data without change.
- No translation code is required to map the in-memory presentation of data to the disk-resident representation, as is done in the file system storage paradigm.
- It is easy to convert applications to use persistent objects.

The example in this article, discussed in the following section, focuses on many of the aforementioned aspects of a C++ based ODBMS.

MIGRATING AN EXISTING APPLICATION TO AN ODBMS

The example we have chosen to illustrate the relative ease of converting an existing application using an ODBMS is given using ObjectStore for OS/2 from Object Design, Inc. It is based on a code sample from an simple application that maintains a database of phone calls. (ObjectStore for OS/2 is the data management component of this C++ application written with C Set ++ from IBM and the OS/2 environment.) This setup is diagrammed in Figure 1.

The original database, maintained in a flat text file, consists of data representing the total number of phone calls (**volume**) based upon the area code (**ac**) and the long distance carrier (**ld**). The function of the program is to:

- Read in the database and create a binary tree representation,
- Read a line of input and update the correct node in the tree, and
- Write the tree structure back out to the flat file database.

The program consists of three modules—update, read, and write. Figure 2 shows the main program, which constructs the in-memory data structure, reads input from a file, updates the correct node, and writes the translated data structure back to the flat file database. The read module (Figure 3) reads in the flat file database and creates the binary tree structure. The write

module (Figure 4) transforms the tree structure to the disk-resident representation and writes it back to the flat file database. Each node of the tree is an instance of the class `node`, defined in `node.hh` (Figure 5).

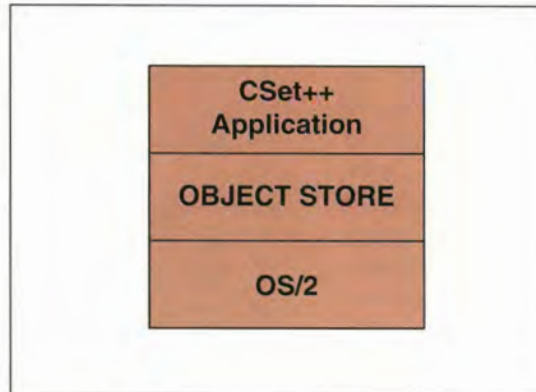


Figure 1. *Component technology*

The original application (Figures 2-4) is self-explanatory to C++ developers and won't be elaborated on here.

When this program is converted to work with ObjectStore (Figure 6), the binary tree structure can

An existing application can be extended to a full object-oriented database management system.

be written directly to the ObjectStore database, thereby eliminating the need for the read and write modules. That is, the binary tree structure is made persistent.

THE MIGRATED APPLICATION

Any application using ObjectStore functionality must first call the static member function `objectstore::initialize()` (line 4 of Figure 6). To create a persistent object, the database in which the object is to be stored must be specified. Before a persistent object is created, therefore, the database must be opened by calling the static member function



database::open() (line 5 of Figure 6). With ObjectStore, every read from or write to persistently allocated memory must be within a transaction. The transaction macros have the form

```
OS_BEGIN_TXN (identifier, exception**,
              transaction_type) and
OS_END_TXN (identifier)
```

In Figure 6, all persistent reads and writes are within the transaction `t1`.

An object that has been stored in persistent storage is retrieved in subsequent processes by looking up the topmost (or entry-point) object by name and then by navigating to the target object. Line 7 of Figure 6 retrieves a pointer to the database root named `root` and line 8 returns a pointer to the entry point object associated with the specified root. The program core (that is, statements within the scope of the while clause) remains unchanged from the original in Figure 1. The database is closed at the end of the transaction by calling the member function `db->close()` (line 10 of Figure 6).

Thus, the additional lines of code required to make this an ObjectStore application consist of calls to:

- Open the database,
- Start a transaction,
- Find the entry point for the node data structure,
- End the transaction, and
- Close the database.

BENEFITS OF MIGRATION

The original application consisting of Figures 2, 3, and 4 is made up of 40 lines of C++ code. The migrated application, shown in Figure 6, contains only 28 lines, a reduction of 30%. In addition to substantial savings in coding, an ODBMS adds multiuser concurrency to a single-user application, transaction consistency with restart/recovery, and database administration capabilities.

After migrating your C++ application, you can address areas of improved functionality. For example, you might wish to maintain versions of objects and

```
#include <stdio.h>
#include "node.hh"

extern node* read_graph (char *);
extern node* write_graph (node *, char *);

main (int, char **argv)
{
    node *root, *ac_ptr, *ld_ptr;
    FILE *fp;
    int ac, ld, volume;

    /* ----- Read in the flat file & create the binary tree --- */

    root = read_graph (argvU1e);

    /* ----- Read in the input & update the correct node in the tree -- */

    fp = fopen (argvU2e, "r");
    /*
    Core of the program. Walks a binary tree & updates
    an existing node in the tree.
    */
    while (fscanf(fp, "%d %d %d", &ac, &ld, &volume) != EOF)
    {
        ac_ptr = root;
        while (ac_ptr->value != ac)
            ac_ptr = ac_ptr->sibling;
        ld_ptr = ac_ptr->child;
        while (ld_ptr->value != ld)
            ld_ptr = ld_ptr->sibling;
        ld_ptr->volume += volume;
    }
    fclose (fp);

    /* ---- Destroy the tree & save the file ----- */

    write_graph (root, argvU1e);
}
```

Figure 2. Original C++ main program

have the capability to merge them together at a later date if needed. Bidirectional relationships and collections facilities can be added to improve application performance and reliability.

Of course, object databases vary significantly in their functionality and performance. When evaluating them, keep the following considerations in mind:

- Performance in your application

environment

- Ease of use and programming
- Ease of migrating existing applications
- Stability and track record of the vendor.

Since the migration example above was based on the ObjectStore ODBMS, we will address some specifics of this database.



```
#include <stdio.h>
#include "node.hh"
node *read_graph(char* infile)
{
    int ac, ld, volume;
    node *root, *ac_ptr, *ld_ptr;
    FILE *fp = fopen (infile, "r");
    ac_ptr = NULL;
    while (fscanf(fp, "%d %d %d", &ac, &ld, &volume) != EOF)
    { /* ----- Create the area code level nodes ----- */
        if (ac_ptr == NULL) {
            ac_ptr = new node(ac);
            root = ac_ptr;
        }
        else {
            ac_ptr = root;
            while ((ac_ptr->sibling != NULL) && (ac_ptr->value != ac))
                ac_ptr = ac_ptr->sibling;
            if (ac_ptr->value != ac) {
                ac_ptr->sibling = new node(ac);
                ac_ptr = ac_ptr->sibling;
            }
        }
        /* ----- Create the long distance carrier nodes --- */
        ld_ptr = ac_ptr->child;
        if (ld_ptr == NULL) {
            ld_ptr = new node(ld);
            ac_ptr->child = ld_ptr;
            ld_ptr->volume = volume;
        }
        else {
            while ((ld_ptr->sibling != NULL) && (ld_ptr->value != ld))
                ld_ptr = ld_ptr->sibling;
            if (ld_ptr->value != ld) {
                ld_ptr->sibling = new node(ld);
                ld_ptr->sibling->volume = volume;
            }
        }
    }
    close (fp);
    return root;
}
```

Figure 3. Read the flat database file and create tree

OBJECTSTORE

ObjectStore is a second-generation object-oriented database management system. Second-generation object-oriented database systems are based on C++, which provides most of the mech-

anisms needed to describe and manipulate persistent objects. Complete database functionality is obtained by additional language and runtime features. ObjectStore databases can also be accessed by applications written in C.

The product augments the language and runtime system to provide facilities for modeling objects and managing large sets of objects, including:

- A collection facility (sets, lists, and so on),
- Queries on collections,
- Relationships between objects, and
- Versioning of objects in support of groupware.

ObjectStore provides a unified API to persistent and transient data, as well as traditional DBMS features such as persistence, transaction management, distributed data access, and associative queries. It also supports the client/server computation model, in which the server handles all access to ObjectStore databases, including storage and retrieval of persistent data. The server is responsible for concurrency control and recovery; transactions involving more than one server are coordinated using the two-phase commit protocol.

ObjectStore client libraries provide an interface between the user's application and the server. The client manages the logical view of data including collections, queries, versions, transaction management, memory management, and relationships among objects. The client environment includes the ObjectStore Cache Manager, which maintains a cache of objects accessed by all the client processes on a machine. (A single cache manager handles the caches of multiple applications.)

ObjectStore can address a database larger than the address space of the operating system by dynamically assigning portions of address space to correspond to portions of the databases used by the application. It provides three programming interfaces: C and C++ library interfaces and an extended C++ that provides a tighter language integration to the query and relationship facilities.

IBM and OBJECTSTORE

In 1993, Object Design and IBM signed a multi-year strategic relationship for developing and using object database technology, with IBM purchasing an



equity interest in Object Design. The two companies have since initiated several joint development projects. IBM Programming Systems will incorporate ObjectStore in several commercial applications involving document management, groupware, and software development.

CONCLUSION

Object-oriented database systems hold promise for applications that deal with large amounts of non-record oriented, complex data. These systems integrate object orientation with database capabilities, simplifying the modeling of real-world problems.

ACKNOWLEDGMENTS

Paul Blanco helped in setting up the OS/2 environment and getting the application running on OS/2. John Treadway was instrumental in getting this article out.

Mohammed Anwaruddin, *Object Design Inc.*, 1 New England Executive Park, Burlington, Mass., 01803. Anwaruddin received his M.S. in electrical and computer engineering from the University of Arizona. He is involved in porting ObjectStore to various platforms. Before joining Object Design, he worked on CAD tools and Network Management Engineering at Digital Equipment Corp. While there, he spent a year at the University of California, Berkeley, as a Visiting Industrial Fellow working with Prof. Randy Katz on the design of Version Server. Anwaruddin can be reached on Internet at anwar@odi.com.

```
#include <stdio.h>
#include "node.hh"

void write_graph (node *root, char *outfile)
{
    node *ac_ptr, *ld_ptr;

    FILE *fp = fopen(outfile, "w");
    ac_ptr = root;
    while (ac_ptr != NULL) {
        ld_ptr = ac_ptr->child;
        while (ld_ptr != NULL) {
            fprintf (fp, "%d %d %d\n", ac_ptr->value, ld_ptr->value,
                ld_ptr->volume); | ac_ptr->child = ld_ptr->sibling;
            delete ld_ptr;
            ld_ptr = ac_ptr->child;
        }
        root = ac_ptr->sibling;
        delete ac_ptr;
        ac_ptr = root;
    }
    fclose (fp);
}
```

Figure 4. Write tree structure back to flat file

```
class node {
public:
    int value;
    int volume;
    node *sibling;
    node *child;
    node( int val) {
        value = val;
        volume = 0;
        sibling = child = NULL;
    }
}
```

Figure 5. Class definition file

REFERENCES

- Cattell, R. G. G. *Object Data Management*. Redding, Mass.: Addison-Wesley, 1991.
- Lamb, C., G. Landis, J. Orenstein, and D. Wienreb. "The ObjectStore Database System," *Communications of the ACM*, Vol 34, No. 10, October 1991.
- ObjectStore Technical Overview*, Object Design Inc.
- Zdonick, B.S. and D. Maier. *Readings in Object-Oriented Database Systems*. San Mateo, Calif.: Morgan Kaufman Publications, 1990.

FINALLY, CLIENT/SERVER INTEGRATION.

Not long ago, client/server development required massive amounts of time, money and expertise to combine different and complex technologies.



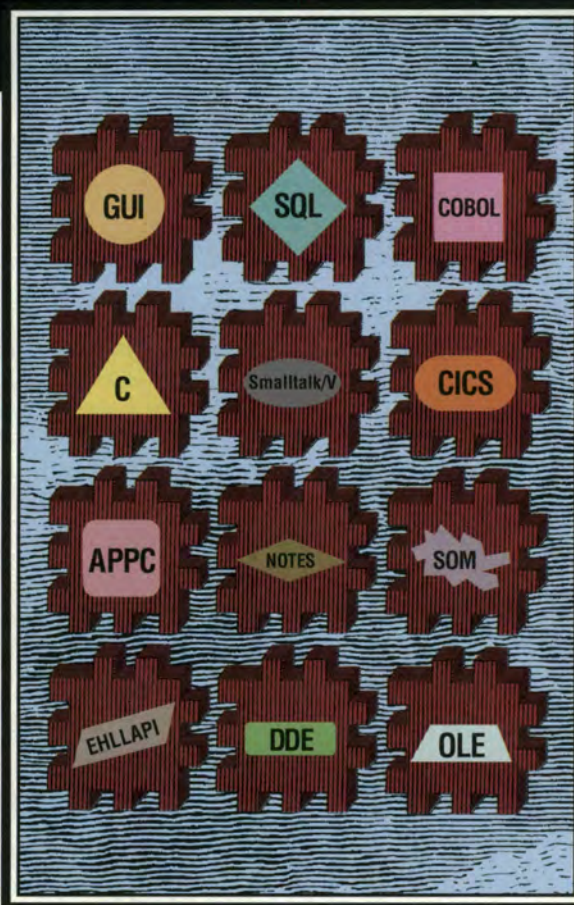
Now Digitaltalk PARTS®, a rapid application development tool set, lets you easily integrate your software assets into

client/server applications.

PARTS is the only object-oriented technology that lets you leverage your legacy code and the knowledge of your current staff.

Only PARTS products let you take existing code—written in Smalltalk/V, COBOL, C, SQL and other languages—and wrap it into components or “parts.” Which can then be virtually snapped together visually. The result is smooth-running client/server applications in a fraction of the usual time. For a fraction of the usual cost.

PARTS supports all popular SQL databases like Sybase, Oracle and DB2. Plus legacy or late model



systems like CICS, COBOL, APPC and SOM. And PARTS lets you develop on both OS/2 and Windows.

RATED #1—TWICE.

Only months ago, PC WEEK awarded PARTS Workbench the highest rating ever in the OS/2

category, calling it “the definitive visual development tool.”

And InfoWorld ranked PARTS the #1 component-based tool for visual development. InfoWorld's Stewart Alsop adds: “There’s nothing like it on the PC.”

To make large teams productive, PARTS also supports group development and version control. Plus PARTS has a host of graphical power tools to give you all the power of objects—without the learning curve.

10 YEARS EXPERIENCE.

And PARTS is from Digitaltalk. The company that’s been providing object-oriented tools to the Fortune 500 longer than anyone else in the world—with over 125,000 users.

Call 800-531-2344 X 606 and ask about our

PARTS Workbench Evaluation Kit.

With minimum effort, you’ll learn why PARTS is the maximum solution for client/server integration.



PARTS. THE CLIENT/SERVER INTEGRATION TOOL.

DIGITALTALK



```
#include <ostore/ostore.hh> /* 1. Objectstore include file */
#include <stdio.h>
#include "node.hh"

database *db; /* 2. Objectstore declaration */
main (int, char **argv)
{
    database_root *db_root; /* 3. Objectstore declaration */
    node *root, *ac_ptr, *ld_ptr;
    FILE *fp;
    int ac, ld, volume;

    /* ----- Initialize ObjectStore ----- */
    objectstore::initialize(); /* 4. ObjectStore statement */

    /* ----- Open the OO database ----- */
    db = database::open(argvU1e); /* 5. ObjectStore statement */

    /* ----- Read in the input & update the correct node in the tree -- */
    fp = fopen (argvU2e, "r");

    /* Core of the program. Walks a binary tree & updates
    an existing node in the tree. */

    OS_BEGIN_TXN (t1, 0, transaction::update); /* 6. ObjectStore start Transaction */
    db_root = database_root::find("root", db); /* 7. ObjectStore: find entry point */
    root = (node *)db_root->get_value(); /* 8. Objectstore: Pointer to entry point */

    while (fscanf(fp, "%d %d %d", &ac, &ld, &volume) != EOF)
    {
        ac_ptr = root;
        while (ac_ptr->value != ac)
            ac_ptr = ac_ptr->sibling;
        ld_ptr = ac_ptr->child;
        while (ld_ptr->value != ld)
            ld_ptr = ld_ptr->sibling;
        ld_ptr->volume += volume;
    }
    OS_END_TXN (t1); /* 9. Objectstore: end the transaction */
    fclose (fp);
    db->close(); /* 10. Objectstore: Close the ODBMS */
}
```

Figure 6. Migrated main C++ program

PVCS 5.1
Now Available

Jim Besemer
Director of Research & Development
INTERSOLV

"WE BUILT THREE VERSIONS OF OUR APPLICATION TO RUN ON EIGHT PLATFORMS. THERE'S NO WAY WE COULD HAVE GOT THAT PRODUCT OUT WITHOUT SOFTWARE CONFIGURATION MANAGEMENT. I'D USE PVCS EVEN IF I DIDN'T WORK HERE."

If you're facing an assignment as tough as Jim's, you're ready for The PVCS Series.

If your development team has more than one member, your application has more than one module or you're working on a LAN—you need automated software configuration management.

The PVCS Series is your best choice for LAN-based Software Configuration Management (SCM). We'll help you automate manual processes and avoid the headaches that multi-version, multi-module applications can cause.

You can skip the lost data, the bug fixes that don't stay fixed, the builds that include outdated modules, overwritten code and the frantic midnight phone calls about wrong versions that somehow made their way into production.

Instead of wasting time wrestling with your development environment, you can concentrate on building quality applications, regardless of what language, programmer workbench, methodology or operating system you're using.

PVCS eliminates tedious housekeeping details and automates builds so they are consistent and error free.

PVCS gives you the security of project-wide undo and the confidence to create good code, knowing you've got a complete audit trail to show where you've been—and help plan where you're going.

Operating seamlessly across Windows, NT, MS-DOS, OS/2 and a variety of UNIX environments, PVCS is flexible enough to fit your development style and the most complex new applications.

The PVCS Series is the market leader in SCM for the LAN. It's a complete family of products, designed for the widest variety of distributed client/server development architectures and operating systems.

The PVCS Series includes:

PVCS Version Manager, PVCS Configuration Builder, PVCS Production Gateway, PVCS Developer's Toolkit and PVCS Reporter.

Put the power of The PVCS Series to work for you. Call for a free demo disk, evaluation, or a copy of the whitepaper, "Software Configuration Management: Choosing The Correct Interface".



The PVCS Graphical User Interface makes it easy to apply the power of PVCS in your distributed development environment.

CALL 800-547-PVCS EXT. 40

SOFTWARE CONFIGURATION MANAGEMENT FOR HETEROGENEOUS LAN-BASED DEVELOPMENT

INTERSOLV



The authors' previous article on custom controls ("A Color-Full Example: Using Color In Control Design," OS/2 Developer, September/October 1993) discussed how to create a threaded control. This article shows how to utilize the control within a Workplace object. **BY CHRIS ANDREW, MARK A. BENGE, and MATT SMITH**

An Object of Many Colors: Using Custom Controls Within A Workplace Object



Chris Andrew



Mark A. Bengé



Matt Smith

In our previous article, we presented a threaded control that could be used to pick colors similar to those within the color palette of the Workplace Shell. In this article, we will show you how to use this control within a Workplace Shell object in order to create a Workplace class that you can utilize. Source code can be found on the usual electronic bulletin boards and on CompuServe, as detailed in the References section.

The color wheel control we developed will now be utilized within a Workplace Shell object class, namely a color palette. This class is meant to illustrate some of the principles of object-oriented programming, such as inheriting behavior from a parent object class, while at the same time illustrating that there is nothing magical about object-oriented programming or the Workplace Shell when integrating your own controls or those of the system.

The new color palette provides a replacement object class for the system-provided color palette, with some enhancements to the original design. It shows how you can write other derivatives of the system-provided WPPalette object class to create an icon, bitmap, or other application-defined palette. Once you have grasped the simple concepts behind Workplace Shell programming, you should have no trouble creating your own object classes.

We will also package our Workplace Shell object class into another custom control that can be used in any application, not just within the confines of the Workplace Shell itself.

A BRIEF INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Object-oriented programming is a powerful technique that allows applications to be developed in separate encapsulated units that frequently correspond to real-world problems modeled within the software. Each unit is known as an object class, and the functions that can be performed by an instance of that class are known as methods of the class. The object class also defines a data structure, which only the methods are allowed to manipulate, that exists for every instance of the object class.

In other words, an object is merely a data structure that is manipulated exclusively by a discrete set of functions associated and instanced with that object's class, like that shown in Figure 1. The definition of the data structure and the functions that operate upon it is provided by its object class. Functions that manipulate an object are known as its methods; the variables within the data structure are often referred to as instance variables.

The creation of a new object involves little more than allocating and initializing a new data structure to represent the state of that object. Since the format of the data structure is defined by the class, the method used to create a new object (sometimes known as a factory method or, in C++, as the constructor) is always provided by the class. Similarly, the destruction of an object (the destructor in C++) amounts to the freeing of that data structure once all the system resources that it utilizes have been cleaned up properly.



The only other thing you need to understand about object-oriented programming is how the principle of inheritance works—that is, how an object class can be written that inherits behavior from another object class. In the explanation up to this point, we have mentioned that the state of an object is uniquely described by a data structure contained within it. That data structure actually contains all the instance variables for all of the classes from which this object class is descended. Similarly, the object supports all the methods defined by all of the classes from which it is descended, in addition to whatever new methods it has defined. The key concept is that when an object class is defined it is able to add new methods and override the existing methods defined by its parent classes.

INVOKING A METHOD ON AN OBJECT DERIVED FROM TWO PARENT CLASSES

Think of this in the same way as subclassing a window in PM. Obviously, after a window has been subclassed it still responds to the same PM messages it did before. However, there is no guarantee it will respond in the same fashion to those messages. The subclass procedure is able to override the processing of any message and can also add processing for new user defined messages.

In PM there is no limit to the number of times a window can be subclassed, provided it is done properly. Each subclass is able to add or remove message processing from all previous subclasses of the window. It is also able to keep its own state variables for a window that has been subclassed in order to control its behavior.

Object-oriented programming allows us to do exactly the same thing, but instead of adding or removing processing for a message we are modifying the behavior of a particular method of that object class.

Now that we've briefly outlined the concepts of what happens behind the scenes in object-oriented programming, we can briefly discuss IBM's System Object Model and the way it has been used in OS/2 2.x.

THE SYSTEM OBJECT MODEL (SOM)

Most object-oriented programmers make use of an object-based programming language, such as C++

Unraveling the Great SYS_DLLS - OS2.INI File Mystery

Something that has been a mystery to many OS/2 developers due to lack of documentation is how to register a custom public window class, thereby making your control public to all applications. Public window classes are only part of this mystery; we will discuss it along with other details concerning SYS_DLLS.

There are 3 keynames defined for the SYS_DLLS application name: Load, LoadPerProcess, and LoadOneTime. Within the OS2.INI file, the strings associated with these keynames are blank-delimited lists of DLLs that are loaded—and whose ordinal 1 (ord1) function is called—at certain times, depending upon which SYS_DLLS list the DLL name appears. The ord1 function for a specific DLL is defined in the .DEF file, like that shown in Figure 2, used to build the DLL. If you want to make your control public, the ord1 function would contain the WinRegisterClass call, which implements the CS_PUBLIC class style illustrated in Figure 3; we will go into more detail on this later.

For the Load keyname, the ord1 function is called each time WinCreateMsgQueue is called. Load is used if you need to do a thread-level initialization like setting a thread-level exception handler. Although it used to be the only option, with the advent of other loading options it is of limited usefulness.

For the LoadPerProcess keyname, the ord1 function is called for the first WinCreateMsgQueue in a process. LoadPerProcess can be used if you have, for example, a common heap. The first time your DLL is loaded, register your window classes and allocate a sharable heap; on all subsequent loads, you need only access the shared memory.

For the LoadOneTime keyname, the ord1 function is called once and only once, when PM is initialized by the shell. LoadOneTime can be used to register a public window class.

Registering A Public Class. To register a public window class for your control, add your DLL to the LoadOneTime list. Your DLL, in turn, will contain an ord1 function that performs the WinRegisterClass call, which uses the CS_PUBLIC style shown in Figure 3. Remember, CS_PUBLIC is only valid when used within the shell process. Therefore, to make your control public, perform the following steps:

1. Make a backup copy of your current OS2.INI file.
2. Add your DLL to the LoadOneTime list for SYS_DLLS in the OS2.INI file.
3. Shut down and reboot.

We have included a sample application that shows how to add your DLL name to the SYS_DLLS LoadOneTime list. (Note that if CS_PUBLIC is used by any process other than the shell process, WinRegisterClass will fail.)

Restrictions. The appname SYS_DLLS, and the keynames Load, LoadPerProcess, and LoadOneTime are all case sensitive; you must be aware of the case sensitivity as shown in the sample application. Additionally, there is a 255 character limit per each keyname string. Finally, if you have modified the string for the LoadOneTime keyname, you must reboot for your change to take effect.

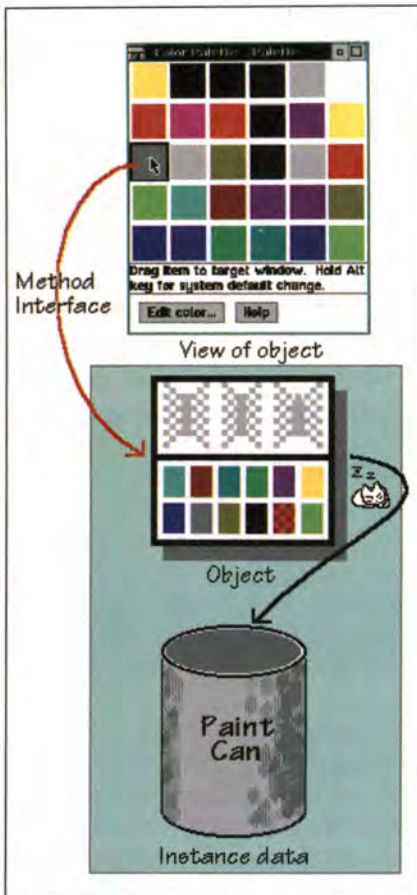


Figure 1. Invoking a method on an object

or Smalltalk, to develop object-oriented applications. These languages normally come with a large library of predefined object classes that can be used as a starting point for deriving new object classes.

When the Workplace Shell for OS/2 2.0 project was started, there were no object-oriented languages capable of generating native 32-bit code. Given the object-oriented user interface of the shell, it seemed likely that the shell would have to be developed without the benefit of any object-oriented programming structure beneath the covers.

Fortunately, the System Object Model (SOM) technology that was being experimented with in the IBM AIX environment came to the rescue. SOM is really just a protocol that allows object-oriented classes to be defined in any programming language, even a procedural language such as C.

```
LIBRARY    YOURDLL

DESCRIPTION 'Your Controls DLL'

EXPORTS

    yourOrd1Function    @1

    *
    *
    *
```

Figure 2. Ordinal definition within definition file

```
WinRegisterClass (    (HAB)NULL,
    "YourPublicClass",
    (PFNWP)YourWindowProc,
    CS_PUBLIC,
    sizeof(ULONG) +
    sizeof(PVOID));
```

Figure 3. Defining public class

The elegance of SOM comes in its ability to create a language-independent definition of the interfaces to a particular object class, one that can then be used in any programming language. Thus it would be possible to develop an object-oriented application using SOM in which the object classes that comprised the application could be implemented in different languages.

In OS/2 2.x, a SOM run-time support DLL basically handles all the nasties of creating and destroying objects, making sure that the object classes conform to SOM programming conventions. In the OS/2 2.x toolkit there is a SOM compiler that will take a language-independent definition of an object class (the .CSC file) and convert it into a set of skeleton functions and header files that can be used with IBM C Set/2, IBM C Set++, WATCOM C/C++32, Borland C++ for OS/2, or Zortech C++ for OS/2 compilers.

WORKPLACE SHELL PROGRAMMING

The Workplace Shell is implemented as a series of SOM object classes. The definitions of these classes are shipped with the OS/2 2.x toolkit as a series of

class definition files (.SC files in the \Toolkit2\SC subdirectory) and their corresponding C syntax bindings (contained in .H files in the \Toolkit2\C\OS2H subdirectory).

A common misconception is that SOM is used only with Workplace Shell programming. SOM is available for use with or without the Workplace Shell. It is important to note, however, that SOM (which comes with OS/2 2.x) does not yet provide any useful object class libraries of its own. It is really best used from another object-oriented programming language such as C++ that has its own set of ready-to-run class libraries and books full of sample programming techniques.

Workplace Shell is run automatically when the OS/2 operating system is started. The object classes that comprise the shell provide for the creation and destruction of persistent objects (objects that stick around even though they are invisible to the user) which can be stored as files or directories on the hard disk or, alternatively, stored in the OS2.INI file as binary data blocks.

The Workplace Shell also uses the container and notebook controls to display objects to the user as icons; the

WITT: Instant record,

Now there's a faster, easier, more affordable way to test your applications. Introducing Workstation Interactive Test Tool (WITT) 2.1, for OS/2® 2.0. An AD/Cycle product from IBM Programming Systems.

WITT does it all. Record, edit, playback and screen compare.

*Introducing
Workstation
Interactive
Test Tool 2.1*

So once you've built a test case, you can use it over and over again.

It works for applications on OS/2 PM, and host-connected MVS, VM, VSE and OS/400®.

Now regression testing is more reliable. New code testing is easier.

• And learning is easier, too. With a friendly online tutorial.

instant replay,



The development team at IBM Santa Teresa created WITT 2.1 to help you improve the quality of your applications.

What's more, WITT won't test your budget. Because it's very affordable.

It only takes an instant to call 1 800 426-3346, ext. 64, and ask for more information and our free evaluation demo. Or, better yet, order WITT today with a no-charge two-month testing period and developer hotline assistance.

• The instant you get WITT 2.1, you'll be more productive.

instant relief.





Method	Purpose
<code>wpSetupCell</code>	This method allows the data contained within a particular cell of a palette object to be modified. Note that because the palette handles storage of the cell data automatically, any changes made to the cells of the palette are persistent.
<code>wpRedrawCell</code>	Forces a particular cell within the palette to be redrawn. Usually used in conjunction with the <code>wpSetupCell</code> method in order to allow the user to see changes in the palette as they are made.
<code>wpPaintCell</code>	A palette subclass must override this method in order to display a graphical representation of the data contained within a given cell of the palette.
<code>wpQueryPaletteInfo</code>	Returns the current properties of the palette object, such as the number of rows and columns of cells and the size of the each cell element in pels.
<code>wpSetPaletteInfo</code>	This method may be called at any time to modify the number of cells within the palette or to set the dimensions of each cell displayed. Note that if the cells are of variable width or height, the maximum height and maximum width must be set using this method, or cell data could be clipped.
<code>wpEditCell</code>	When the user clicks on the Edit... push button at the bottom of the palette or double clicks on one of the palette cells, this method is invoked to bring up a value editing dialogue. Once again, all subclasses of <code>WPPalette</code> ought to override this method and invoke a suitable dialogue.
<code>wpQueryPaletteHelp</code>	This method should be overridden to tell the palette which panel to display when the user requests help (by pressing the Help push button in the palette window).
<code>wpclsQueryEditString</code>	Returns the text displayed in the Edit... push button in the palette view.

Table 1. Description of methods defined by *WPPalette* object class

properties of those iconic objects are displayed in a settings notebook. In addition, the shell provides a set of defined method-based programming interfaces for the definition of the context menu, settings pages, object details to be displayed in Details view, and so on.

One of the most important features of the Workplace Shell is that it is easily customized by the user, but still very extensible. Since even the system-provided object classes can be modified to suit the purposes of the user, the shell is truly an open-ended application.

To customize the Workplace Shell, you must write one or more SOM object classes, placing them within a DLL. You also have to register, using `WinRegisterObjectClass`, those object class-

es with the shell so they can become part of the shell's list of available object types.

Because of the object-oriented design of the Workplace Shell, you don't have to get involved in the usual PM programming chores such as creating a message queue, variable storage, initialization, or using advanced features such as the container control, notebook control, or drag-and-drop. That functionality has already been written into the Workplace Shell; the associated behavior is inherited by all object classes. The only thing you must do is write the new behavior of your object class.

As illustrated in the following example, it is possible to create a new object class that is implemented by fewer than

a dozen lines of C source code. All its basic behavior is automatically inherited from its parent Workplace Shell object class.

The Workplace Shell provides an object class called `WPPalette` that provides nearly all the functionality required to implement our `ColorPalette` object class. All instances of `WPPalette` are capable of storing a variable number of data blocks, known as cells. Whenever the value contained within a particular data block gets modified, the palette object automatically saves that change back to its persistent form. In other words, we don't have to worry about how or where to store the colors to be displayed by our palette.

`WPPalette` also provides a view that shows a graphical representation of the



Figure 5. An instance of the ColorPalette class opened in Palette view

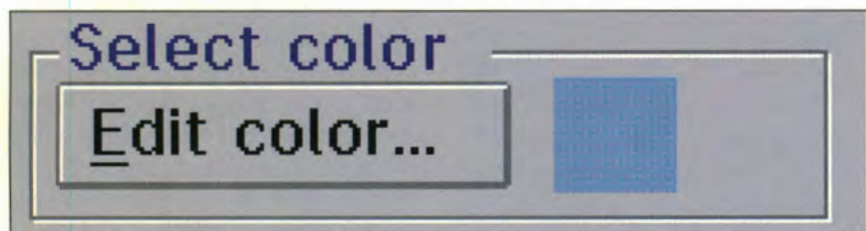


Figure 6. Color picker dialogue containing color wheel custom control

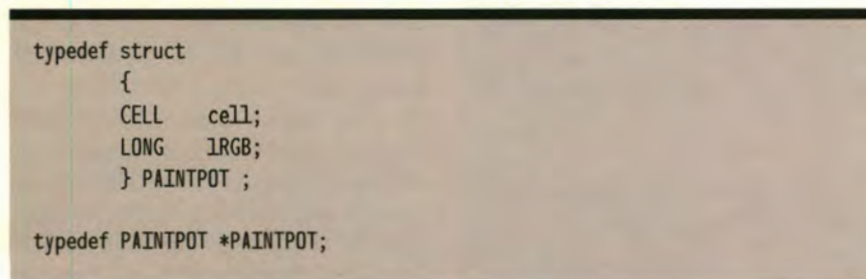


Figure 7. Cell data used within a color palette

value of each data block. The palette manages the creation of the window, scrolling and positioning the cells to be displayed. The only code we have to write is a routine that displays the value contained by a given data block and tells the palette how much room is required to draw each particular cell (the cells can be of different sizes). The palette view, shown in Figure 5, allows the user to drag a value from a cell so it can be applied to another window and also to edit the value of a given cell within the palette. We will completely override the method that edits a cell, displaying our own dialogue. This dialogue, shown in Figure 6, contains the color wheel custom control that lets the user pick a color. We will allow the existing drag-and-drop functionality to be handled by our parent object class, as it needs no modification. Table 1 summarizes the purpose of the various new methods defined for WPPalette.

In actuality, we will not be deriving our object class from the WPPalette class but rather from the system-provided WPColorPalette class. We wish to inherit as much function as possible from our parent class, and WPColorPalette is the closest match to our desired functionality. Don't believe that explanation? Okay, would you believe that we're just plain lazy?

The WPColorPalette object class contains all the code we will need to handle dragging and dropping colors from our palette. Each cell within an instance of the WPColorPalette object class is actually a PAINTPOT structure in Figure 7. The first element of the cell data must always be the CELL structure (contained in the wppalet.h header within the \ToolKt21\C\OS2H subdirectory), which merely contains a field describing the size of the whole structure in bytes. The rest of the structure can contain anything (in this case, just an RGB value), but notice that each cell's data does not have to be the same size.

Now that we have discussed the basic functionality of the system-provided color palette, we can consider what methods we need to override in



Hints And Tips From Von Den Workplace Hackmeister (Or, Confessions Of A Workplace Junkie)

- When developing a class, you may get tired of watching the Workplace Shell open all the windows you had open the last time. The automatic restart function can be disabled using the following options in your CONFIG.SYS file:

SET RESTARTOBJECTS=NO

- We all make mistakes in our code, but the Workplace Shell is frequently very forgiving of serious mistakes: its exception handler just carries on to the next method call. The trouble is that this can be a hindrance during the development of an object class, when you'd love to know where your code is malfunctioning. Setting the following environment variable in your CONFIG.SYS file will turn off the exception handler:

SET SHELLEXCEPTIONHANDLER=NO

Many people experience problems when trying to compile and test an object class DLL. This is because the Workplace Shell loves to hang on to your object class DLL and keep it loaded until all instances of your object class have disappeared. There is, however, usually no need to reboot your system. The following hints can be helpful:

- Register your object class only once. The shell remembers that you have registered it, so you waste time if you de-register and register every time you modify your DLL.
- Create only one instance of your object class for testing purposes; put it in a folder that can be closed, so the usage count for your class DLL will be either one (if the object is visible) or zero (if the folder containing it is closed). When the usage count for your DLL reaches zero, the shell will usually unload the DLL unless your object has class associations to a filetype or file extension.
- To ensure that the Workplace Shell unloads your DLL as quickly as possible, use the following environment variable in your CONFIG.SYS file:

SET OBJECTSNOOZETIME=0

- If these techniques aren't working for you, consider using a small program as a replacement for the Workplace Shell, such as the MYSHELL.EXE program provided with the source code. This allows you to start and stop the shell just like any other application. When it comes time to try another version of your DLL, just stop the shell, replace the DLL, and then restart the shell.
- We recommend kernel debugging to debug your object class, as you will be better equipped to understand when and how your methods are called because you have symbolic information for both the Workplace Shell DLL (PMWP.DLL) and the SOM.DLL (hey, we titled this "confessions of a hacker," didn't we?).
- If you are planning to use the kernel debugger, you will probably want to use the external prefix and class prefix operators in your .CSC files when you define your object class. This

(continued on page 54)

our subclass, which we'll call **ColorPalette**. Since all object classes ought to identify themselves with a unique icon and also name themselves so the user can easily distinguish which category of object he or she is dealing with, it is advisable to override both the `wpclsQueryIconData` and `wpclsQueryTitle` meta-class methods when creating a new Workplace Shell object class.

Also, the `wpclsInitData` method has been overridden, so we are guaranteed to have registered the color wheel control on the Workplace Shell process using the `WinRegisterClass` API. This is done because the `wpclsInitData` method, called only once when the class is loaded, is an ideal place to perform one-time initialization and window class registration. Otherwise, there is no real need to override any of the other standard Workplace Shell methods and we can concentrate on the palette methods described in Table 1.

For a start, we wish to override the `wpEditCell` method, so our object class will exercise the color wheel control created in our previous article. Overriding this method will merely load up our `ColorPickingDialog` using `WinLoadDlg` if it hasn't been previously loaded. The user will then be able to modify colors in the palette using the color wheel control. Notice that the `ColorPickingDialog` is non-modal, providing the easiest way for the user to change values of multiple elements of the palette very quickly. The user is free to choose another palette cell to be edited while leaving the `ColorPickingDialog` displayed.

In addition, we are going to override the `wpPaintCell` method to demonstrate how to draw the cell data. This isn't strictly necessary, since our parent class `WPCOLORPALETTE` already draws each `PAINTPOT` as a circular splotch of color. It is, however, useful in illustrating the drawing model used within palette objects. To paint each cell, we will draw a filled polyline in its particular color using the `GpiBeginPath`, `GpiMove`, `GpiPolyline`, `GpiEndPath`, `GpiSetColor` and `GpiFillPath` APIs. The shape of the polyline figure to be drawn will be defined by a

For OS/2 2.x

UNLEASH THE POWER OF VISPRO/REXX™

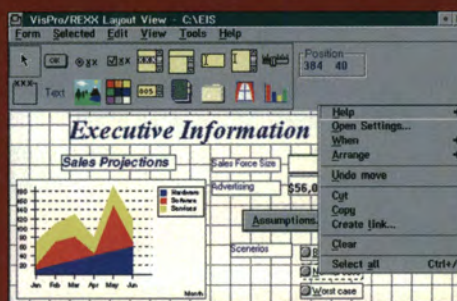


**Available
and Ready**™

for OS/2

VisPro/REXX is an easy-to-use visual programming environment that gives you the power to create your own OS/2 2.x GUI applications. Fast.

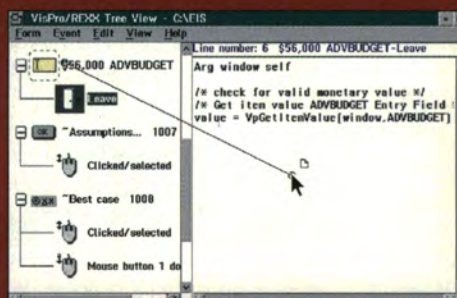
VisPro/REXX features the inherent advantages of Presentation Manager, a Workplace Shell based Project Manager, the Common User Access (CUA) '91 user interface and the SAA REXX programming language.



**Design
Your Layout**

VisPro/REXX offers easy access to OS/2 programming features, including buttons, lists, sliders, charts, business graphics, DB2/2, APPC and EHLLAPI. The upcoming SOM toolkit will allow you to add GUI objects.

Best of all, you don't have to be a REXX expert to use VisPro/REXX: the drag and drop programming feature creates the REXX statements for you.



**Drag and
Drop to
Build Your
Program**

With VisPro/REXX, you can quickly develop OS/2 CUA '91 applications, build client server programs and generate a single .EXE file of your masterpiece for royalty-free distribution.

UNLEASH YOUR POWER TODAY!

\$299

\$99 Bronze Edition

System Requirements:
OS/2 2.x, 5Mb memory and
1.5Mb hard disk space.

HockWare, Incorporated
P.O. Box 336, Cary, NC 27512-0336
Fax (919) 380-0757 Phone (919) 380-0616

**Hock
Ware™**

Putting You in Control

VisPro/REXX is a trademark of HockWare, Incorporated.

OS/2, DB2/2, CUA, SAA and Workplace Shell are registered trademarks of International Business Machines Corp.
© 1993 HockWare, Incorporated. All rights reserved.



new method for our object class, which we'll call `QuerySampleShape`.

When the user double clicks the mouse pointer on the square of color or clicks the mouse pointer on the Edit push button, the override `wpEditCell` method created earlier is called. Through this method, we communicate

allows—after the information has been saved within a private dialogue structure—for user selection of the color wheel to be sent back to the color palette once the notification message has been received from the color wheel within the dialogue.

function you wish to add. As an example, consider the code in Figure 8 for an object class derived from the `ColorPalette` object class. All this derived class does is override the `QuerySampleShape` method so the palette cell entries are drawn as star shapes instead of rectangles of color. Using the provided source code, try creating other simple subclasses of the `ColorPalette` object class. This will allow you to get familiar with the tools and processes needed to create Workplace Shell object classes.

REXX INSTALLATION PROGRAM

Once our object classes are written, the final step is to install them into the Workplace Shell and create some instances of those classes. This can be done through the `WP Class List` object shipped with the OS/2 toolkit, found in the PM Development Tools folder contained in the Toolkit 2.1 or Toolkit 2.0 folders. It is, however, worth illustrating how easy it is to register and create object classes using a simple REXX command file, as shown in Figure 9.

Notice that the REXX procedure is able to specify various parameters to the palette objects using the setup string parameter. This technique is very similar to passing control data to a PM control window as it is created; however, instead of a complex structure, the object setup string is just a series of `key=value` pairs separated by semicolons. All classes of Workplace Shell object support at least a few setup string keywords. For more examples, study the file `INI.RC` (installed on any OS/2 2.x machine in the `\OS2` directory on the boot drive). This script file is used to create all standard objects that can be seen after OS/2 installation.

COLOR SAMPLE CONTROL WINDOW CLASS

Now that we've created a color palette object class that exists within the confines of the Workplace Shell application process, we will create a simple window control that allows that palette functionality to be accessed from any application program in the system.

The average OS/2 user is already

(continued from page 52)

will cause SOM to make each method in your class a non-static function, starting with the prefix you specify. This saves a lot of pain because SOM creates default static functions for each method, leaving you without symbolic information for your methods.

- If you can't stomach the thought of working in assembler all the time without the benefit of your source code, you can use the IBM PM Debugger (IPMD). To use the IPMD source level debugger, you will need to run it as your Workplace Shell process. This is easily achieved by changing the `RUNWORKPLACE` environment variable in your `CONFIG.SYS` file:

```
RUNWORKPLACE=IPMD.EXE
```

- When you reboot your machine, the debugger will now come up in place of the Workplace Shell. You can now start, stop, and restart the Workplace Shell process at will. The program to run is `PMSHELL.EXE` in the `OS2` directory.
- Setting breakpoints in your object class code can appear near-impossible at first glance. However, you will find that setting a load breakpoint on the DLL that contains your object class—when `PMSHELL.EXE` has been loaded, but before you run it—works nicely. When you hit the load breakpoint for your DLL, go ahead and stick breakpoints in your code on the stuff you're trying to debug. Remember that when you are tracing through code that makes method calls, it does not call you method directly. This can make source level debugging quite painful because you don't have source for SOM itself, the code that figures out what method to call. This trick is to not trace into a method call, but rather to stick a breakpoint on the method override that you've added. Alternately, through IPMD, you can step debug into the method. The button on the tools bar is:



- Debugging an object-oriented program is very much like debugging a PM application. That is, your method overrides and new methods can be invoked at any time by the system or from your own code. If you really want to know what is going on, try sticking a breakpoint on each and every method and override in your class to see when and how it is used.
- Finally, if you at first you don't succeed, reboot your machine and try again. It has been known to fix many a problem or two or three or...

with the `ColorPickingDialog` using a private message `MSG_EDITCELL`.

This message is given the current cell color and cell information, which allows it to set the cross hairs of the color wheel to the current color within the color palette swatch. It also

STARCOLORPALETTE SAMPLE OBJECT CLASS

To get back to a point made earlier, it is possible for an object class to inherit the behavior of every method defined by its parent class and write a very small amount of code to implement the new

VNR KNOWS OS/2®

**CLIENT/SERVER PROGRAMMING
WITH OS/2 2.1, Third Edition** by Robert
Orfali and Dan Harkey. OS/2 MONTHLY called
the 2nd Edition *"The OS/2 Book of the Year!"*
Others said individual chapters were by
themselves *"worth the price of the book."*
This new edition keeps you state-of-
the-art in OS/2 client/server
developments. \$39.95 0-442-01833-9

**OS/2 2.1 CORPORATE
PROGRAMMER'S
HANDBOOK**

by Nora Scholin, Martin Sullivan
and Robin Scragg. A quick and
easy modular approach to writ-
ing software programs. Gives
code examples for using OS/2
2.1 to its full potential.
\$39.95 0-442-01598-4



**OS/2 2.1 REXX HANDBOOK:
Basics, Applications and Tips**
by Hallett German. A one-stop
guide to one of the most popular
command languages. Combines
a basic tutorial with fast answers
to all REXX questions.
\$29.95 0-442-01734-0

**WRITING OS/2 2.1 DEVICE
DRIVERS IN C, Second Edition**
by Steven J. Mastrianni. Defines each
type of driver and how to fit them into
your system with tips on debugging,
tuning, and enhancing driver
performance. Includes disk with
source code for four complete drivers
plus all source code found in book.
\$39.95 with disk. 0-442-01729-4

Van Nostrand Reinhold

115 Fifth Avenue New York, New York 10003



To place an order, or for a complete listing of VNR titles, call
1-800-544-0550 (Dept. Z 1555) or GO VNR on the CompuServe® Network.

OS/2® is a registered trademark of IBM Corporation.





familiar with the concept of picking a color, font, or scheme from a palette of predefined values. So why not make that function available globally?

When the user pushes the Edit Color... push button (like that in Figure 10) displayed on the color sample control, that control communicates with our Workplace Shell palette object class

cation process where the control lives to the actual palette object (which lives under the Workplace Shell process). The method of creating the setup string to display itself is shown in Figure 11.

It causes the shell to open the palette object in palette view or to resurface an existing open view of the palette. When

decides that the palette doesn't contain a suitable color, he or she just brings the color picking dialogue up to select another color into the palette. Neat, eh?

COMING UP

Beginning in our next article, we will implement a replacement to the list box control. Not only will the list box cir-

```
SOM_Scope BOOL SOMLINK clr_QuerySampleShape(
    ColorPalette *somSelf,
    PPOINTL      pPoints,
    PULONG       pcPoints)
{
    #define STAR_CPOINTS 11
    static POINTL
    ptlStar[STAR_CPOINTS] = { { 20, 0 },
                               { 30, 40 },
                               { 0, 60 },
                               { 30, 60 },
                               { 50, 100 },
                               { 70, 60 },
                               { 100, 60 },
                               { 70, 40 },
                               { 80, 0 },
                               { 50, 30 },
                               { 20, 0 } };

    if ( pPoints )
        memcpy(pPoints, ptlStar,
              STAR_CPOINTS * sizeof(POINTL));
    if ( pcPoints )
        *pcPoints = STAR_CPOINTS;
    return TRUE;
}
```

Figure 8. C source code for StarColorPalette class

so a view of the standard color palette appears. Having this functionality in several applications would allow the user to create a set of favorite colors, each stored within a single color palette—and then have easy access to them from any application. If a user is not satisfied with the colors that exist in the palette, he or she can always bring up the color picking dialogue and revise the color choices.

CONTROL SAMPLE CODE

The sample control will communicate with the Workplace Shell object, using the WinSetObjectData API call to pass a setup string parameter from the appli-

```
SysRegisterObjectClass('ColorPalette', 'CLRPALET');
SysCreateObject('WPFolder', 'Palette Folder',,
    '<WP_DESKTOP>',,
    'OPEN=ICON;OBJECTID=<PALETFLDR>', 'update');
SysCreateObject('ColorPalette', 'Color Palette',,
    '<PALETFLDR>',,
    'OPEN=DEFAULT;OBJECTID=<CLRPALET>', 'update');
```

Figure 9. REXX installation code

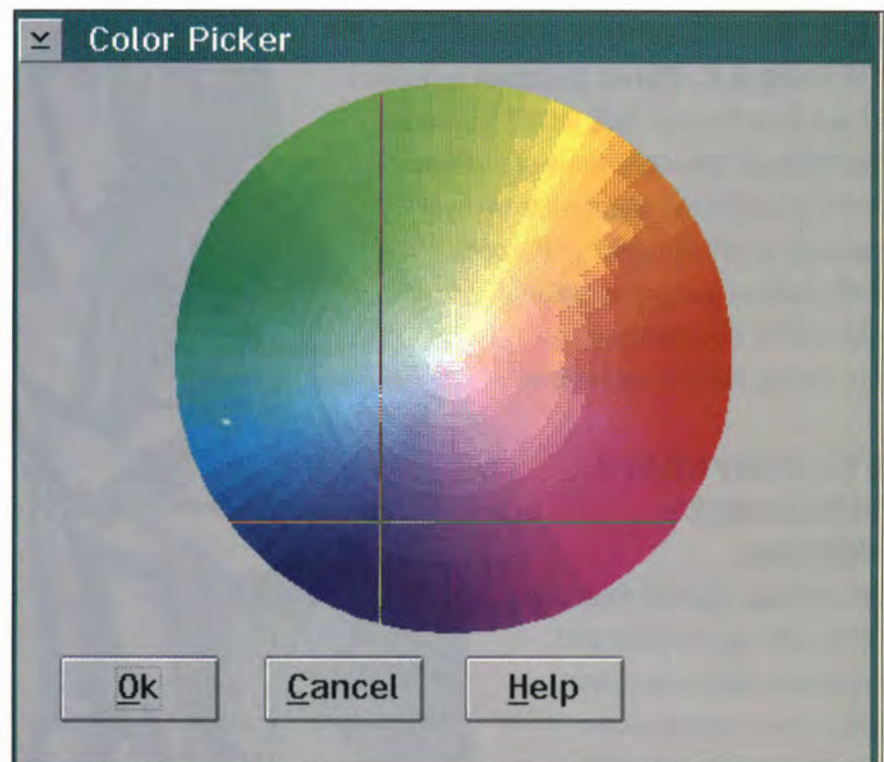


Figure 10. Color sample control

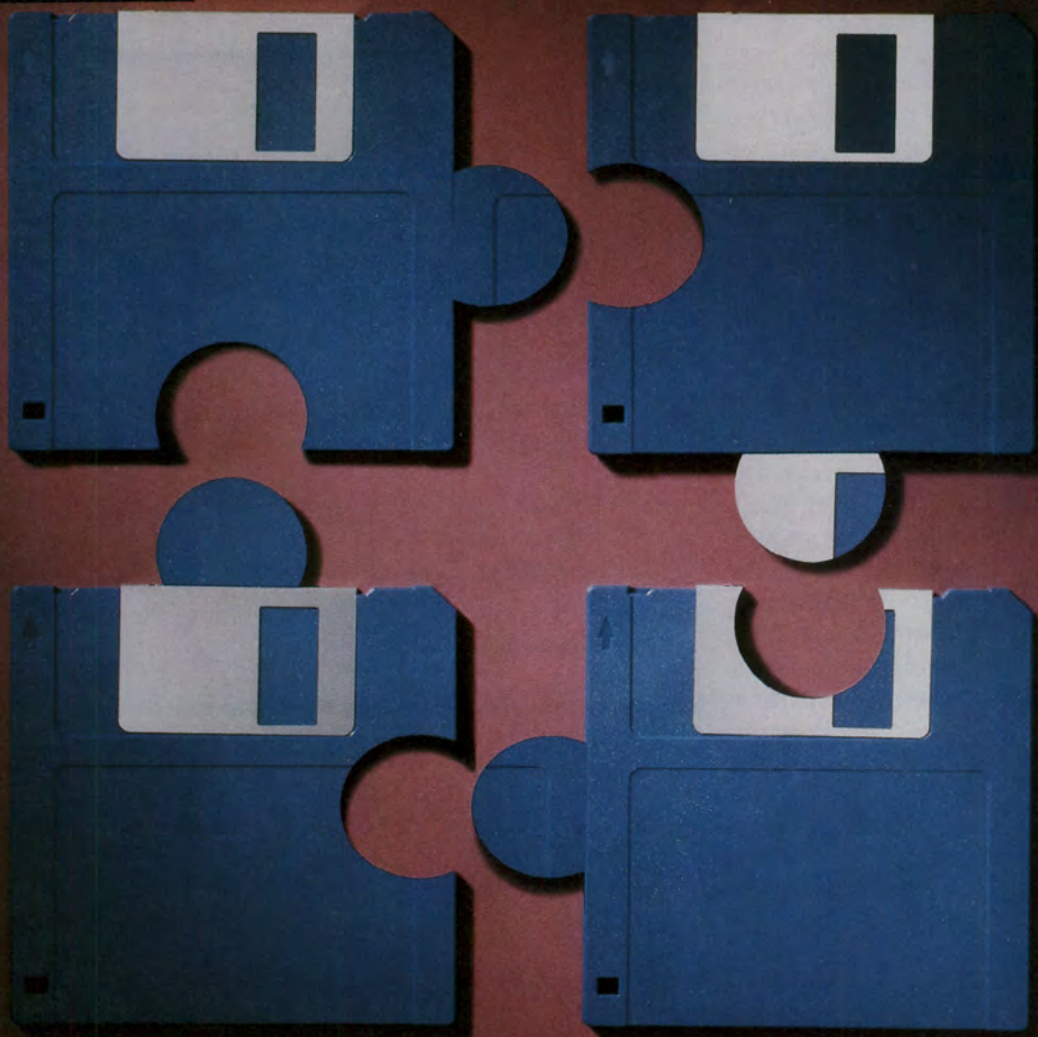
the user picks a color, the palette then sends a notification back to the window handle to inform it of the new color choice.

This allows users to pick colors from different applications using a single palette of colors. If at any time the user

cumvent the 64K limitation, it will be enhanced to allow for direct manipulation of text, columns, and other goodies. So stay tuned over the next few issues while we develop the control into something you can use every day in your own programs.

Relational Database Software

Client/Server



Applications Development

End User Tools

INGRES. The complete OS/2 solution.

Now available on OS/2 - The ASK INGRES Intelligent Database. The very same database that has proved so popular with blue chip organizations and high-end professional users in commerce, industry and Government.

Fully featured, fully portable, fully scalable.

The ASK INGRES Intelligent Database is a leading edge, intelligent, relational database, ideally suited to client/server applications. ASK INGRES supports all major desktop and UNIX networking standards as well as IBM SNA, enabling communication with DB2 and IMS.

The ASK INGRES Intelligent Database is complemented by a range of professional-standard rapid application development tools. ASK/Windows4GL - portable, object-oriented, fully featured and

GUI-based - combines intuitive user friendliness with unparalleled power and functionality. ASK/VisionPro packs equally impressive performance into character-based application development and includes such features as automatic code generation. Users can also access the database using ASK/Query and Reporting tools or tools from industry-leading software vendors. In addition, users have the pick of an ever-growing number of third-party ASK INGRES-based applications.

For a free INGRES OS/2 Solutions Kit, please call 1-(800)-446-4737 or 1-(510)-769-1400 (outside North America).

 **ASK Group**





Chris Andrew, WordPerfect Corp., 1555 N. Technology Way, Orem, UT 84057. Andrew works in the OS/2 Shared Code group at WordPerfect Corp. He previously worked for IBM Hursley (U.K.) and IBM Boca Raton. He has been involved with OS/2 Presentation Manager since the OS/2 1.2 release and most recently was a member of the OS/2 Workplace Shell team on the 2.0 release. Andrew has an honors degree in physics from Imperial College of Science and Technology, London.

Mark A. Benge, IBM Programming System Laboratory, 11000 Regency Pkwy., Cary, NC 27512. Benge is a staff programmer who joined IBM in 1989 and has since worked on various CUA '91 controls for OS/2 2.x and CUA Controls Library/2. He now works in IBM class library user interface development, where he is involved in the implementation of C++ classes for direct manipulation. Benge received a B.S. in computer science from Western Carolina University.

Matt Smith, Prominare Inc., 100 Front St. E., Toronto, Ont., Canada M5A 1E1. Smith is lead architect for the Prominare Development System, an OS/2 2.x advanced GUI development environment. He has been actively involved with OS/2 since 1988 and co-founded Prominare in 1990. Smith has a degree in architecture from the University of Waterloo.

```
CHAR    szSetup[100];
HOBJECT hObject;

/* Print our window handle into the setup string so that the
 * palette can communicate back to us
 */

sprintf(szSetup, "OPEN=DEFAULT;CTRLHDL=%lx", (ULONG)hwnd);

/* Send the setup string through to our installed instance of
 * ColorPalette object class
 */

if ( (hObject = WinQueryObject("<CLRPALET>")) !=
    (HOBJECT)NULL )
    WinSetObjectData(hObject, szSetup);
```

Figure 11. Object data setting code

REFERENCES

OS/2 2.0 Programming Guide Volume II (IBM Doc. S10G-6494)

OS/2 2.0 Presentation Manager Programming Reference Volume II (IBM Doc. S10G-6265)

The source code can be downloaded from several electronic sources:

- In the U.S., call the IBM PCC BBS at (404) 835-6600. The source code for this article is in File Area 11 under the name WPOBJ.EXE.
- In Europe, you can find the source code on the International OS/2 Users Group BBS at +44 (0)454 633197 or +44 (0)454 633420.
- On CompuServe, the source code is in LIB13 of the OS2DF2 forum.
- If you have a VM account on an IBM node, you can receive the source code with the command REQUEST WPOBJ.EXE FROM BANZAI AT CARVM3.



The new *OS/2 Magazine* is specifically designed to fit the needs of today's OS/2 users. Feature articles are written by technical experts throughout the OS/2 community. Here are some of the sections that can be found in *OS/2 Magazine*:

New Products . . .

will describe the latest hardware and software for OS/2 buyers and users.

Under the Hood . . .

takes readers inside their OS/2 system, offering suggestions for gaining the ultimate in performance and compatibility.

Customization Alley . . .

shows OS/2 users how to automate their workflow with OS/2's built-in customization tools, like REXX.

Sneak Preview . . .

gives OS/2 product buyers a first-hand look at hot new products through short reviews written by *OS/2 Magazine's* editors and regular contributors.

What's Happening . . .

shares the latest industry news and views with *OS/2 Magazine's* readers -- from insights into IBM strategy to the emergence of new standards.

OS/2 Administration . . .

offers assistance to busy systems administrators, whether volunteers or part of a corporate IS department.

Tips 'n Tricks . . .

will offer a monthly fare of reader-submitted fixes, patches, shortcuts, workarounds, questions, answers, ready-to-use REXX programs, and more!

Don't miss a single issue. This magazine is for you. Subscribe today!

.....
Subscribe now to OS/2 MAGAZINE and receive a savings of 25% off the regular rate.

☒ **YES!** Please start my one-year (12 issues) subscription to *OS/2 Magazine* for just \$29.95. That's a savings of 25% off the regular rate of \$39.95. I pay no money now and agree to pay \$29.95 upon receipt of my invoice. I understand that if I am ever dissatisfied with my subscription, for any reason, I may cancel my subscription by simply writing "cancel" on the invoice and I'll owe nothing.

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Allow up to 6-8 weeks for delivery of your first issue. Basic rate for one year is \$39.95. Canadian/Mexican subscribers please add \$10 for surface mail; overseas add \$40 for airmail. Prepayment drawn in U.S. dollars on a U.S. bank for foreign orders.

**Mail your order to: OS/2 MAGAZINE, P.O. Box 56664, Boulder, CO 80323-6664 or
CALL (800) 765-1291 or FAX (415) 905-2233**



This article describes how to write Workplace Shell-compliant applications in C language, exploiting the API prototyped in PMWPH. **BY STEFANO MARUZZI**

Writing WPS-Compliant Applications

There are now two kinds of OS/2 applications: basic porting from the 16-bit world and the newer 32-bit SOM-integrated development. While the latter is the best way to write reusable code, it requires a deep redesign of applications to fully exploit object-oriented technology. The guidelines offered in this article are an extract from *OS/2 2.1 Workplace Shell Programming* by Stefano Maruzzi.

One of the distinctive features of OS/2 2.1 is the adoption of an object-oriented interface called the Workplace Shell (WPS), a complex multi-threaded OS/2 application automatically executed as the system boots. WPS exploits the standard Presentation Manager (PM) API to create a complete window, window context menus, and other structural components. In addition, objects provided by the System Object Model (SOM) of OS/2 enrich the WPS interface. SOM is a tool to develop, package, and distribute objects among applications. Every object class defined by WPS derives on a class, `SOMClass`, the ancestor of all objects in the system. WPS objects are placed in separate DLLs and developed according to the standard mechanism adopted for either EXE or DLL modules in the traditional PM programming scheme.

Once a DLL is created, it is necessary to register it in WPS. The API function `WinRegisterObjectClass()` takes care of this operation, instructing WPS to load that specific DLL at every following boot. Similarly, `WinDeregisterObjectClass()` removes an object class from the WPS internal list. Almost everything found in the OS/2 System folder is an object (more specifically, an instantiation of an

object). The Color Palette, the Font Palette, and the Templates folder are objects based on WPS classes.

Once opened, objects take the form of windows on the screen. From a user's perspective, there is no way to distinguish between a WPS object and an application window. The title bar contains the window title, while the resizing icons to the right of the title bar let the user either hide or maximize the window. This is the result of mixing SOM objects and PM window classes.

WHAT WPS IS

Upon careful examination, WPS is basically a subclassed window of class `WC_CONTAINER`. The parent window, a traditional frame without any frame control, is always hidden behind it. The frame's parent is the desktop, as shown in Figure 1.

As stated earlier, one of the most publicized feature of OS/2 2.1 is its object-oriented interface (as compared to the application-oriented interface of MS Windows and OS/2 1.x). In Windows, drag-and-drop operations always start from File Manager where this undocumented logic is implemented. In WPS, in contrast, every object present in the shell is a valid source and target of a drag operation, with no preliminary discrimination. You can drag a document to either the printer or the shredder. The shredder itself can be dragged to a second level folder (a folder inside the desktop folder), and so on.

From these direct manipulations, one can get the impression that drag-and-drop operations are system-wide. Although if you analyze WPS behavior from a user perspective this judgment is cor-

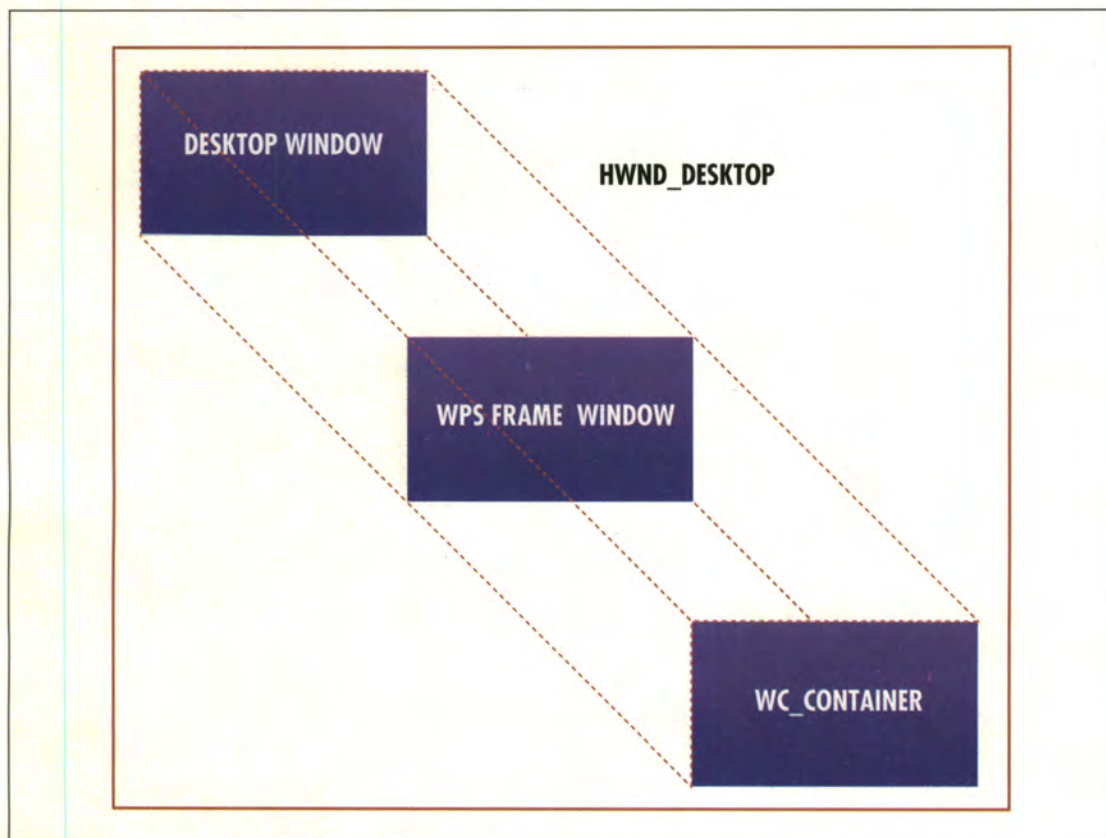


Figure 1. Relationship between the desktop and WPS

```
hwndFrame = WinCreateStdWindow( HWND_DESKTOP,  
                                OL,  
                                &flFrameFlags,  
                                WC_CONTAINER,  
                                szWindowTitle,  
                                CCS_AUTOPOSITION | CCS_EXTENDSEL,  
                                NULLHANDLE,  
                                OL,  
                                &hwndClient) ;
```

```
// set the frame window as the owner  
WinSetOwner( hwndClient, hwndFrame) ;
```

Figure 2. A client that is a window of one of the fifteen predefined window classes

rect, from a PM developer's point of view, the focus is quite different. WPS, despite its role in an OS/2 system, is basically a traditional PM application with a frame window of class `WC_FRAME` and a client class, `WC_CONTAINER`. Everybody can achieve this goal by simply using `WC_CONTAINER` as the class

name in `WinCreateStdWindow()`. In Figure 2, the identifier `hwndClient` refers to the container window.

The net result is the absence of a window procedure under developer control. The whole flow of messages generated by the user interactions directly reaches the `WC_CONTAINER` window procedure; this



Notification	Code	Value Description
CN_DRAGAFter	101	Sent to the owner after the receipt of a DM_DRAGOVER in the container with CA_ORDEREDTARGETEMPHASIS or CA_MIXEDTARGETEMPHASIS, and with a display mode of Name, Text or Detail.
CN_DRAGLEAVE	02	Sent to the owner after the receipt of a DM_DRAGLEAVE.
CN_DRAGOVER	103	Sent to the owner after the receipt of a DM_DRAGOVER and when the container does not have the attribute CA_ORDEREDTARGETEMPHASIS and is in the display mode Tree or Icon.
CN_DROP	104	Sent to the owner after the receipt of a DM_DROP.
CN_DROPHELP	105	Sent to the owner after the receipt of a DM_DROPHELP.
CN_ENTER	106	Sent to the owner after the pressing of the Enter key, or after a double-click with the selection button.
CN_INITDRAG	107	Sent to the owner when a dragging operation is initiated.
CN_EMPHASIS	108	Sent to the owner when a record attribute changes in the container.
CN_KILLFOCUS	109	Sent to the owner when the container loses focus.
CN_SCROLL	110	Sent when scrolling takes place.
CN_QUERYDELTA	111	Sent to get additional information when a scrolling is being executed.
CN_SETFOCUS	112	Sent to the owner when the container acquires focus.
CN_REALLOCPSZ	113	Sent to the owner before the editing phase of one of the container's items terminates (CN_ENDEDIT).
CN_BEGINEDIT	114	Sent when a text string is being changed in the container.
CN_ENDEDIT	115	Sent to the owner when the editing phase of one of the container's items terminates.
CN_COLLAPSETREE	116	Sent after a tree structure is collapsed in the tree view display mode.
CN_EXPANDTREE	117	Sent after a tree structure is expanded in the tree view display mode.
CN_HELP	118	Sent to the owner after the receipt of a WM_HELP.
CN_CONTEXTMENU	119	Sent to the owner when the container receives the message WM_CONTEXTMENU.

Table 1. List of all notification codes for the WC_CONTAINER class

is why the WPS client is subclassed. The detour in the flow of messages reaches first a window procedure inside the application and then the class window procedure where the

container logic resides.

If you consider WPS basically a WC_CONTAINER window, the judgment about the object-oriented interface changes drastically. All supposedly

system-wide operations instead take place inside the same container window or in a parent-child relationship. According to the WC_CONTAINER naming convention, an object is a record (that

OS/2 PRODUCTS & SERVICES

TCP/2™

TCP/IP FOR OS/2

Supports *all* released versions of OS/2
Including 32bit API for OS/2 2.0 and above

Network access from protected, real and WINOS2 sessions
Full server operation for **telnet, ftp, rsh** and **sendmail**
IP gateway capability

DEVELOPERS KIT with socket library

VT102 & 3270 emulation for remote login

TCP/2™-OS/2 TRANSPORT MODULE...\$200.

For further information contact: Essex Systems, Inc.

One Essex Green Drive • Peabody, MA 01960

(508) 750-6200

BUILD YOUR OWN OS/2 LIBRARY

Reprints are now available for **OS/2 DEVELOPER**.

Use your customized reprints for: **Customer support, sales tools, marketing support, educational tool.**

Call today for a custom quote: Laura Pullen, **OS/2 DEVELOPER**
415-358-0126 ext.302 or fax 415-358-9749

SUBSCRIBE TODAY!

Only \$39.95 for a full year subscription to the premier source of tools and techniques for the OS/2 software developer: **OS/2 DEVELOPER**

Call today and don't miss a single issue:

1-800-WANT-OS2

Opt-Tech Sort/Merge

High Performance Sort/Merge/Select Utility

Use as a stand-alone utility or call as a Subroutine. Many features including unlimited file size, multiple keys, record selection, duplicate elimination, summing and more!

Call for free brochure Opt-TechData Processing
P.O. Box 678

Available for: Zephyr Cove, NV 89448
OS/2 or Unix \$249 Phone (702) 588-3737
DOS or Windows \$149 Fax (702) 588-7576

John C. Dvorak's Favorite

"The best of the best...Hobbes OS/2 CDROM... Lots of nifty utilities, fonts, and other goodies on this CD." -John Dvorak, PC Computing, Oct. 93.

Hobbes is 2800 files from the largest ftp archive of OS/2 material, ftp-os2.cdrom.com, quaintly known as Hobbes. Hobbes serves as the major internet distribution site of OS/2 freeware, shareware, device drivers, program updates and information. We bring this entire archive to you in the well organized, indexed, and virus scanned Hobbes disc. Many programs are installed on the CDROM, so you can run them *without* using any of your hard disk space.

The Hobbes OS/2 CDROM is updated quarterly and last updated in October 1993. Start enjoying this disc now. We guarantee your satisfaction.

\$29.95 Many more products available.

Walnut Creek CDROM • 4041 Pike Lane, Ste D-98 • Concord, CA 94520

1 800 786 9907

+1 510 674 0783 FAX: +1 510 674 0821 info@cdrom.com

GET AN EDGE ON YOUR MARKET

With customized consulting services for OS/2 & client server applications:

- Programming
- Project Management
- System Analysis
- Maintenance
- Design
- Testing
- Installation

**REAL-TIME
TECHNOLOGIES**

**(800) 441-9241
(708) 441-9240**

1001 Greenbay Rd, Winnetka, IL 60093

Indelible Blue, Inc. offers a convenient source for all of your OS/2 needs - powerful 32-bit applications, useful utilities, informative books & videos, OS/2 shareware, OS/2 T-shirts - you'll find it all just a phone call away. We strive to maintain the most complete and up-to-date collection of OS/2 products to found anywhere. Look on your local BBS for our on-line catalog, or call us for a print catalog. You'll find top-quality applications from familiar names, such as IBM and Lotus, as well as many others from lesser-known foreign and domestic developers.

(919) 834-7005 Vox (919) 834-2406 FAX
P.O. Box 31306 Raleigh, North Carolina 27622

Attention Mathematica users:
Everything you need to know to get the most out of your math programming is in

THE MATHEMATICA JOURNAL

Call or fax today for information about subscriptions and special issues:
Phone orders: 415-905-2332
Fax orders: 415-905-2233

is, a block of memory privately handled by the class). This chunk of memory is composed of standard information, integrated by additional data specific for that object.

The classic distinctions in four object categories (folders, applications, data files, and devices) is, therefore, inappropriate from a development point of view. They all are icons (records) inside one container, and the difference between the objects depends on the kind of data hooked to each one (object specific information). Every record in a container has some standard data (a RECORDCORE structure) containing basic information such as the image displayed on the screen and its text. The developer adds to this data new information to better qualify a record. This is exactly what has been done in the development of WPS.

Usually the object-specific data amount to a greater quantity than the size of a standard RECORDCORE structure. All this information is undocumented.

DRAG-AND-DROP OPERATIONS IN A CONTAINER

In every OS/2 demo, you have been shown how easy it is to drag a document on the printer or to destroy it by dropping it on the shredder. All these operations occur inside the same container window and are managed through DM_ messages and CN_ notification codes. OS/2 2.1 offers a set of seventeen messages specifically designed to describe every event related to drag-and-drop. These messages are always sent (they bypass the application message queue, reaching the destination window procedure directly) to the window underneath the cursor hot-spot, and eventually to the target window. The WC_CONTAINER class, like any other predefined window class, has its own set of messages (CM_), window styles (CCS_), and notification codes (CN_). When an event occurs in a container window, the owner is notified via a WM_CONTROL message with the notification code packed in the second short in mp1. Some of the nineteen CN_ notification codes (the set is listed in Table 1) relate

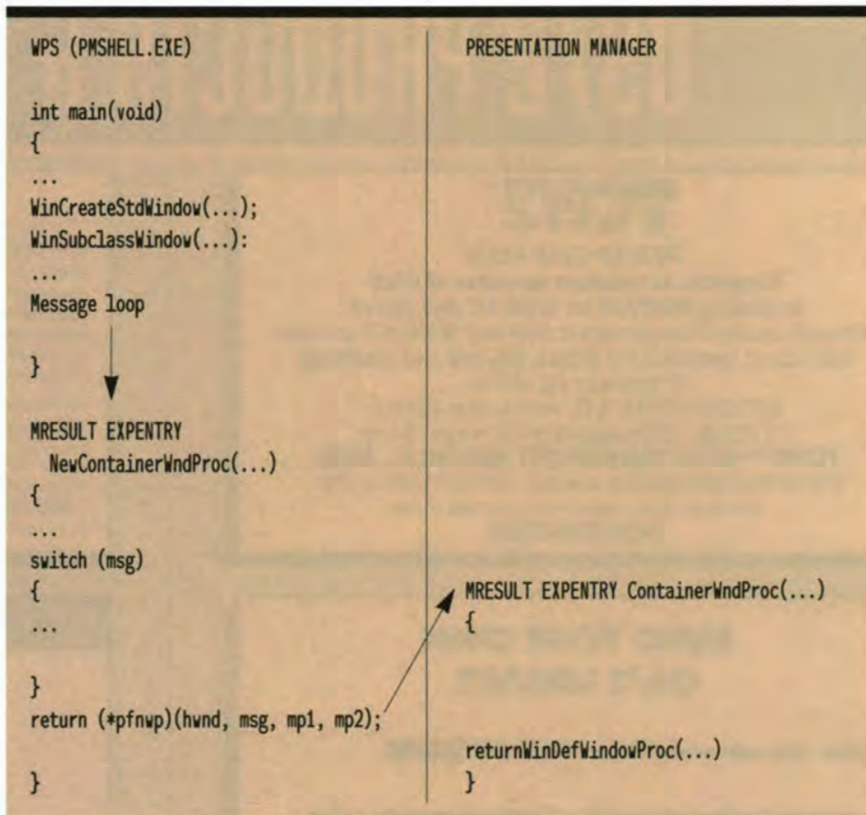


Figure 3. Flow of messages for the WPS application

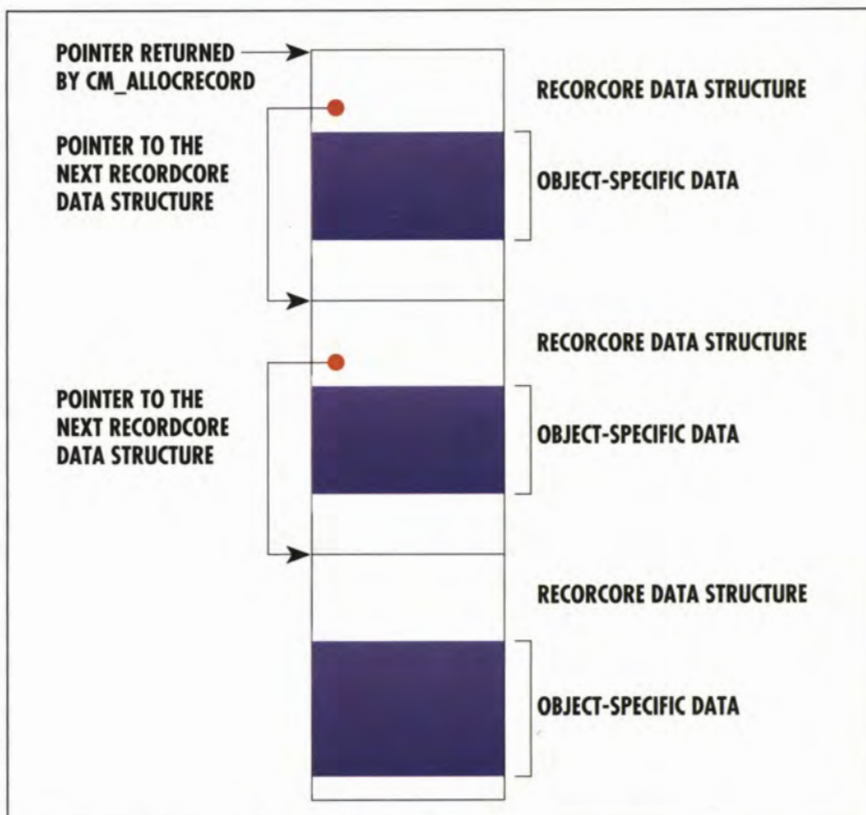


Figure 4. Data associated with each record object in a container window

Mission



IBM Programming Systems introduces
C Set++,™ the most complete application
development package you can buy for
OS/2® Its 32-bit C/C++

compiler lets you unleash
all the power of OS/2 — so you can
create the most advanced, high-
performance applications.

It has an extraordinary code optimizer with a
full set of options. Even a switch to optimize for the new
Pentium™ processor. Plus a full set of class libraries,
including application frameworks for PM, container
classes and classes for multitasking, streams and more.

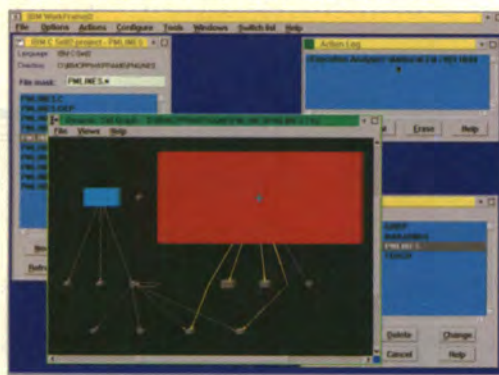
There's also a full complement of other helpful
features. Such as an interactive source level debugger.



C Set ++

shows class library relationships.

What's more, you get Workframe/2,™ a language-
independent tool that lets you customize your own envi-
ronment. It's adaptable and flexible — you can use any 16
and 32-bit DOS, Windows™ and OS/2 tools.



To order C Set++,
contact your nearest dealer or call
1-800-342-6672 (USA) or
1-800-465-7999 ext. 460 (Canada).

Clearly, there's only one place to start. C Set++.

critical code

C Set ++ Technical Features

Standards	ANSI C X3.159-1989
	NIST validated
	ANSI C++ X3J16 (Full ARM)
	ISO 9899:1990
Optimization	Global
	Inter-module
	Function inlining
	Instruction scheduling

starts here.



specifically to drag-and-drop on a container window.

To implement drag-and-drop in a container, it is sufficient to intercept the `DM_` messages received in the target window as a consequence of the call of `DrgDrag()` in the source application or to trap notification codes in the container's owner window procedure. Despite the complexity and the quantity of data involved, in many circumstances drag-and-drop refers to the same container window or strictly related window (another folder with a parent relationship to the WPS client window). Data comes from and reaches the same kind of object (i.e., a container).

Subclassing a `WC_CONTAINER` window offers the opportunity to intercept all `DM_` messages generated during a drag-and-drop manipulation. For WPS, all the `CN_` notification codes reach the frame window procedure. To achieve this goal, you need only remember a simple though intriguing detail. By default, all the frame controls in a standard window refer to the frame window as their owner. This rule, however, doesn't apply to the client window, which has no owner. In WPS, the client's owner is explicitly set to the hidden frame window to intercept all notification codes. This is a wise thing to do when the client is a window of one of the fifteen predefined window classes, as shown in the last portion of Figure 2.

Based on this assumption, let's examine the information that reaches the frame window. Notification codes such as `CN_DRAGINIT`, `CN_DRAGOVER`, and `CN_DROP` contain in `mp2` a pointer to some class-specific data structures like `CNRDRAGINIT` (Figure 5) and `CNRDRAGINFO`. These structures are very simple and have few members.

When the user starts dragging an object inside a container, its owner receives a `CN_INITDRAG` notification code, which collects valuable information in `CNRDRAGINIT`. The second member in `CNRDRAGINIT` is a pointer to a `RECORDCORE` structure (Figure 6), the data type where all record information is stored.

Among the `RECORDCORE` members there

```
typedef struct _CNRDRAGINIT
{
    // cdrginit
    HWND hwndCnr ;
    PRECORDCORE pRecord ;
    LONG x ;
    LONG y ;
    LONG cx ;
    LONG cy ;
} CNRDRAGINIT ;

typedef CNRDRAGINIT *PCNRDRAGINIT ;
```

Figure 5. The `CNRDRAGINIT` data structure

```
typedef struct _RECORDCORE
{
    // recc
    ULONG cb ;
    ULONG flRecordAttr ;
    POINTL ptIcon ;
    struct _RECORDCORE *preccNextRecord ;
    PSZ pszIcon ;
    HPOINTER hptrIcon ;
    HPOINTER hptrMiniIcon ;
    HBITMAP hbmBitmap ;
    HBITMAP hbmMiniBitmap ;
    PTREEITEMDESC pTreeItemDesc ;
    PSZ pszText ;
    PSZ pszName ;
    PSZ pszTree ;
} RECORDCORE ;

typedef RECORDCORE *PRECORDCORE ;
```

Figure 6. The `RECORDCORE` data structure

```
typedef struct _CNRDRAGINFO
{
    // cdrginfo
    PDRAGINFO pDragInfo ;
    PRECORDCORE pRecord ;
} CNRDRAGINFO ;

typedef CNRDRAGINFO *PCNRDRAGINFO ;
```

Figure 7. The `CNRDRAGINFO` data structure

are handles to icons and bitmaps to represent a record (object) on the screen in different views—icon, tree, and detail. This means that an application always knows the object aspect when dragging starts. Furthermore, the developer sets the object-related text

```
typedef struct _DRAGINFO
// dinfo
{
    ULONG cbDraginfo ;
    USHORT cbDragitem ;
    USHORT usOperation ;
    HWND hwndSource ;
    SHORT xDrop ; SHORT yDrop ;
    USHORT cditem ;
    USHORT usReserved ;
}
DRAGINFO ;

typedef DRAGINFO *PDRAGINFO ;
```

Figure 8. The `DRAGINFO` data structure


```
typedef struct _DRAGITEM
{
    // ditem
    HWND hwndItem ;
    ULONG ulItemID ;
    HSTR hstrType ;
    HSTR hstrRMF ;
    HSTR hstrContainerName ;
    HSTR hstrSourceName ;
    HSTR hstrTargetName ;
    SHORT cxOffset ;
    SHORT cyOffset ;
    USHORT fsControl ;
    USHORT fsSupportedOps ;
}
DRAGITEM ;

typedef DRAGITEM *PDRAGITEM ;
```

Figure 9. The DRAGITEM data structure

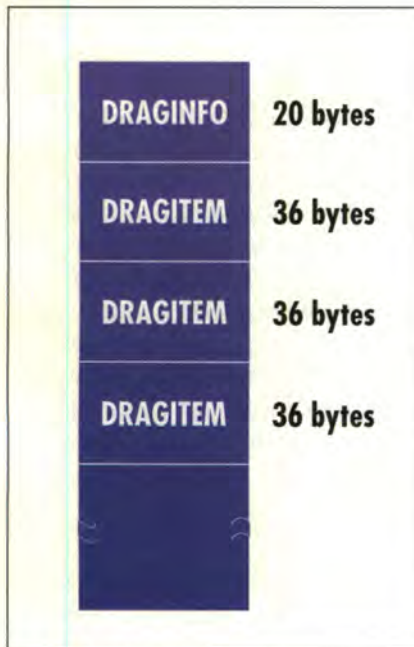


Figure 10. Memory block allocated by an application system before executing a drag

(i.e. the writing underneath the icon on the desktop) in the last three members.

In Figure 7, we examine CNDRAGINFO to understand how this information is passed to the target container (maybe the WPS window itself or an inner folder) after the user releases the right mouse button.

This structure has a member of type PDRAGINFO, a pointer to a DRAGINFO struc-

ture, and a pointer to a RECORDCORE structure. The access to a DRAGINFO data structure provides all information related to a drag-and-drop operation in WPS, as shown in Figure 8.

All data associated with the mouse during the dragging of one or more objects is packed in a memory block of variable size. At the beginning is a DRAGINFO structure, followed by as many DRAGITEM structures as instructed by the cbDragItem member (Figure 9). Figure 10 illustrates how information is stored by the source application before starting a drag process.

Once again, even in the case of a drop onto a container window, WC_CONTAINER information and direct manipulation data are strongly related. The last three HSTR members contain useful information for the target window. The handle to the string in hstrContainerName refers to the source container. Considering that in OS/2 2.1 a folder is the equivalent of a directory in the file system, this means that the physical origin in the file system is always known by the target.

The opposite is never true; right now there is no way for the source window to find out where an object is placed. I'm referring specifically to when an object is either moved or copied between two WPS folders. The feedback provided by WPS is limited to a series of DM_DRAGOVERNOTIFY messages, with mp1 pointing to a DRAGINFO structure slightly modified according to the return value of the DM_DRAGOVER message. The source window of the direct manipulation gets the target window handle as the return value of DrgDrag(). From this point on, I have defined an empirical way to obtain the full pathname of the target destination of a previous drag-and-drop. Once you have a handle, it is easy to query the window title. The first portion of it is the directory name, from which you can jump back to the complete pathname.

To summarize, in the case of a container window (despite the fact that the source and the target containers are either the same or different windows), the information provided through the notification codes (CN_) and direct

manipulation messages (DM_) offer a detailed view of what is happening in the system.

DRAG-AND-DROP OPERATIONS WITH DIFFERENT WINDOWS

Problems arise when you try to exchange information via drag-and-drop with another running application such as an editor. The worst situation is when the flow of information is from a WPS object to an application. In this case, a good portion of the information usually provided through a RECORDCORE data structure is no longer available

Even in the case of a drop onto a container window, WC_CONTAINER information and direct manipulation data are strongly related.

because the target application is not a window of class WC_CONTAINER and the two processes don't share the same address space.

On the other hand, when the source of information is an application and the target is WPS, you need only prepare the DRAGINFO and DRAGITEM structures according to what WPS expects (reverting to DrgDragFiles(), as in the copy of multiple files, doesn't require filling in any DRAGxxx data structure). The internal support to drag-and-drop is limited to container windows, a big disadvantage because one objective is to write WPS-compliant applications that are well integrated in the system shell. The solution to this problem lies in writing specific code inside each application that makes it drag-and-drop aware.

Despite this lack of truly system-wide object management tools, a smart developer can bypass this limitation with a hint of imagination and some experi-



ence. This result can be achieved by writing plain C code and interacting with WPS objects as if they were windows. I always spend a lot of time trying to design an intuitive user interface to my applications; in one case, I designed a kind of panel window that was originally empty. Its surface is divided in two columns and six rows (both dimensions are variable). Each cell of the panel can accept an object represented as an icon. The user selects one object from the desktop and drops it in on empty cell. The result is shown in Figure 11.

The **PANEL** application is basically a valueset window, one of the five new predefined window classes added with the introduction of OS/2 2.x. Like the **WC_CONTAINER** class, **WC_VALUESSET** has some specific notification codes related to direct manipulation. While it is easy to understand that a drag is occurring (a valueset receives the standard **DM_** messages), when the user releases the right mouse button there is no **RECORDCORE** data structure to supply all that valuable information. To obtain the object icon in that case, I refer to **WinLoadFileIcon()**, a new API prototyped in **PMWP.H**. This function loads an application icon from different sources (but doesn't always work properly). This is exactly what WPS does in the General page in Notebook Settings when changing the object icon via drag-and-drop.

The code quality of applications such as **PANEL** would greatly benefit from better-documented WPS behavior. Considering that every program

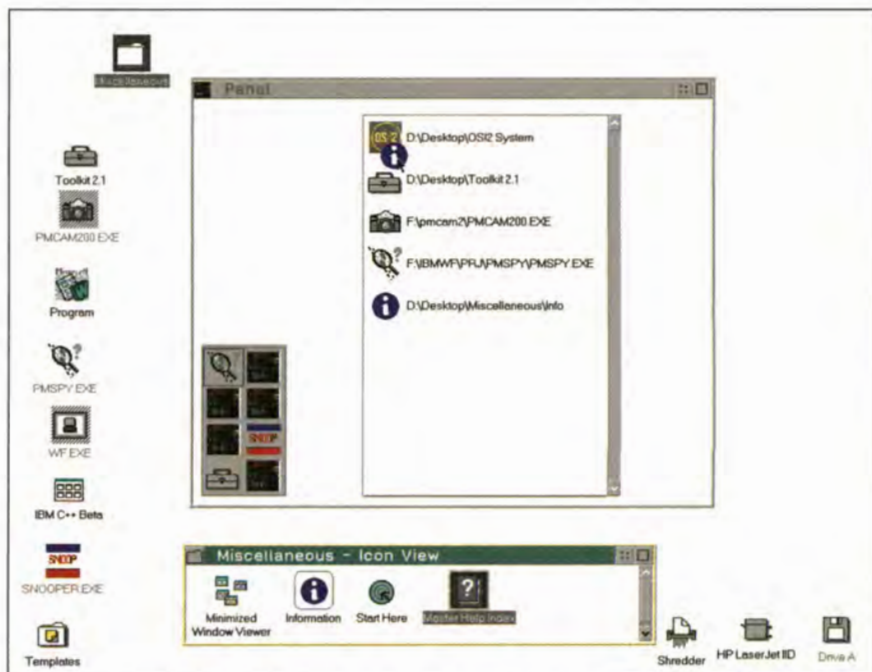


Figure 11. Panel window in which each cell of the panel can accept an object represented as an icon

has to interact with WPS, knowing how the system shell responds to every event is a primary need.

Conclusion

OS/2 2.1 offer a broad range of tools to develop easy-to-use applications. The key components are drag-and-drop support and an extensive use of WPS objects (with the addition of new classes). If developers follow this approach to the use of WPS objects, a set of new and exciting applications will fully exploit the power of OS/2 2.1 in the near future.

Stefano Maruzzi is the author of books on MS Windows and OS/2 programming in Italy. His first book for the U.S. market is *IBM OS/2 2.1 Workplace Shell Programming* (New York: Random House). Maruzzi writes technical articles on WPS programming. He gave a presentation on the subject at the summer 1993 Software Development Conference. Maruzzi holds a degree in agricultural economics from the *Universita' degli Studi of Milan, Italy*. He can be reached on Compuserve at 100115,356 and on MCI Mail as Stefano Maruzzi.

Cut your development time

Quadron's OS/2 development tools team up with IBM

ARTIC co-processor cards to give you flexibility, communication power, and development speed.

Use Quadron software and ARTIC cards to gain extra ports and enhance host PC performance by moving protocol processing to the cards. Develop standard and custom communications for async, BISYNC, HDLC, X.25, LAP-B, or your own special protocol. And if you need more than one protocol, no problem—just run them at the same time on the same card.

Programmers feel right at home with our software. Just program in standard C, and use our messaging API to link OS/2 threads with

tasks on the card. Then check it out with our symbolic debugger and real-time analysis tools in OS/2 windows.

Our easy-to-use software serves applications as diverse as telephone switching, financial transactions, satellite communications, and process control. If any of this sounds like a match for your needs, call us today.

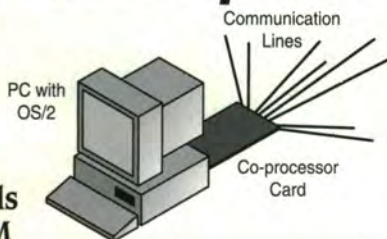


Quadron

209 East Victoria Street
Santa Barbara, CA 93101
Fax: 805-966-7630
805-966-6424



Trademarks are the property of their respective owners.



TLIBTM Version Control For DOS, OS/2 and Windows-NT

• The experts loved TLIB 4:

"...amazingly fast... TLIB is a great system." **PC Tech Journal**
"TLIB has features and power to spare... TLIB is easy to use and the fastest of the reviewed packages." **Computer Language**
"I will not program without it." **Uptime Magazine**

• Now TLIB 5.0 adds:

Automatic branching. Automatic version labeling across branches. User defined **promote** structures, for staged development. Exclusive whole-level **change migration** for customized software. N-way-tree version numbers. Branch and full locking. OS/2 & NT support.

• Plus the features they loved in TLIB 4:

Check-in/out locking. Branching, for parallel development. Keywords. Full binary file support (does not depend upon CRs in the file like other products). Wildcard and list-of-file support; can create lists by scanning source code for includes. Can merge (reconcile) multiple simultaneous changes and undo intermediate revisions. Network and WORM optical disk support. Mainframe-compatible delta generator for Pansophic, ADR, IBM, Sperry formats. Includes integrated PD MAKE by L. Dyer; also integrates with OpusTM MAKE, SlickTM MAKE, others.

MS-DOS **\$139**, OS/2 & NT (with MS-DOS) **\$195** + shipping.
5 station network: MS-DOS \$419, OS/2 \$595. Call for other sizes.



Burton Systems Software

PO Box 4156, Cary, NC 27519 **(919) 233-8128**
FAX: 233-0716

DON'T MISS OUT on a single issue of OS/2



Make checks payable to Miller Freeman, Inc.
Foreign orders must be in U.S. funds. Please allow eight weeks for subscriptions to begin.

Subscription rates for 6 issues:

- ☐ U.S. - \$39.95
- ☐ Canada - \$55.95
- ☐ International surface mail
- ☐ International air mail - \$69.95
- ☐ Check inclosed

Charge my:

- ☐ Visa ☐ MasterCard ☐ AmEx

Card # _____
Exp. Date _____
Signature _____
Name _____
Company _____
Address _____
City _____
State/Zip _____

OS/2 Developer offers tips and techniques for more efficient computing for the advanced software developers who have recognized the need for 32-bit power.

TO ORDER

For new orders & customer service:

1-800-WANT-OS2

(1-800-926-8672)

or

708-647-5960

FAX 708-647-0537

Written subscription orders:

OS/2 Developer

P O Box 1079

Skokie, IL 60076-8079



Futures

This article describes IBM's microkernel technology, examining its architecture and associated programming abstractions. It also describes how the microkernel can be used as the foundation for a highly modular, portable, and secure operating system that can support multiprocessor and multiple operating system personalities.

BY PYLEE LENNIL

The IBM Microkernel Technology

The IBM microkernel technology is based on a Mach microkernel originally developed by Carnegie Mellon University. It also contains selected technologies developed by the Open Software Foundation Research Institute, as well as functionalities implemented by IBM. The major features of the microkernel can be summarized as follows:

- Simplified architecture
- Expandability
- Portability
- Multiprocessor support
- System security.

The microkernel provides a simple, highly modular and extensible architecture compared to a traditional operating system kernel. Extensibility allows many traditionally kernel-based operating services to reside outside the kernel at the user process level. In the traditional operating system, services such as process management, virtual memory management, network management, file system services, and device management are all built into the kernel. This results in a cumbersome and inflexible kernel that is difficult to enhance or port to different hardware platforms. These system kernels also normally support uniprocessor and are inherently less secure. The microkernel, on the other hand, is highly secure. It provides only basic system services such as task and thread management, interprocess communication, virtual memory services, I/O, and interrupt services. These services are made available to user-level tasks through a set of microkernel interface functions. The small kernel also facilitates real-time support, as only a minimum of kernel code must be executed.

Unlike in a traditional operating system, services

such as file system, network services, and device drivers operate outside the microkernel as user-level servers, as shown in Figure 1. The clean separation of kernel servers reduces complexity, increases portability and requires fewer entries into the kernel level to perform operating system functions. In addition, the servers allow the creation of a highly modular operating system that can support multiple system personalities and services by configuring them as user-level servers.

The microkernel is easily portable to different computer platforms due to the clean separation of machine-dependent and machine-independent code. Machine-dependent and device-dependent code is located in the microkernel; porting to a new hardware platform mainly involves rewriting the machine-dependent code.

The traditional operating system kernel is not very secure, with security features built outside the kernel. The microkernel, on the other hand, is designed with security as an integral feature of the system. The microkernel operates on system resource objects such as virtual memory space, files, and processors. User-level tasks access these objects by sending messages through highly secure communication channels called ports.

In a traditional operating system, device drivers operate at the privileged level. Under the microkernel, however, they operate at the user-process level, above the kernel. This means that device drivers can be developed and debugged just like user programs; this also increases portability into other hardware platforms. Since the device drivers are privileged microkernel tasks, they get access to I/O resources such as I/O ports,



memory-mapped devices, and direct memory access (DMA) through the microkernel services. Interrupts are normally intercepted by the microkernel, which decides whether to handle the interrupt or send it to the user-process level device drivers. These drivers must be registered with the microkernel and must maintain interrupt handlers waiting to service the interrupts.

MICROKERNEL ARCHITECTURE

The microkernel provides the following programming abstractions, from which other operating system functionalities can be derived:

Task. A program running under the microkernel is a task. Each task owns a collection of resources such as virtual address space, threads, and ports. The microkernel provides each task with protected access to system resources through ports, communication channels maintained by the microkernel. A task can communicate with other tasks through ports. In a client/server environment, a client task can request services from a server task using interprocess communication. Tasks may be created, terminated, suspended, or resumed by other tasks. When a new task is created, it can inherit virtual address space and port capabilities from its parent task.

Thread. A thread is the smallest independent unit of execution (instruction stream) running within a task. Each task requires at least one—possibly more—concurrently executing threads to perform a computation. Threads share the task address space and all other task resources such as virtual memory, file descriptors, and ports. Threads are scheduled to a processor by the microkernel, which allows multiple threads to run concurrently within a single task's memory space. For example, a multithreaded program may use one thread to compute a mathematical formula while another is used to receive user input. Multithreaded tasks provide programs with greater performance and modularity. In uniprocessor environments the use of threads increases the performance of the task by overlapping the execution of the computation and I/O threads. In a symmetric multiprocessing environment (one in which processors share the same physical memory), the microkernel allows threads in a multithreaded program to execute in parallel on different processors. This

results in substantial improvement in the task's performance.

Objects. The microkernel is object-oriented in nature. All microkernel services and resources such as tasks, threads, virtual memory regions, files and processors are represented as objects. Each object provides a set of operations that can be invoked by sending messages to that object, which in turn

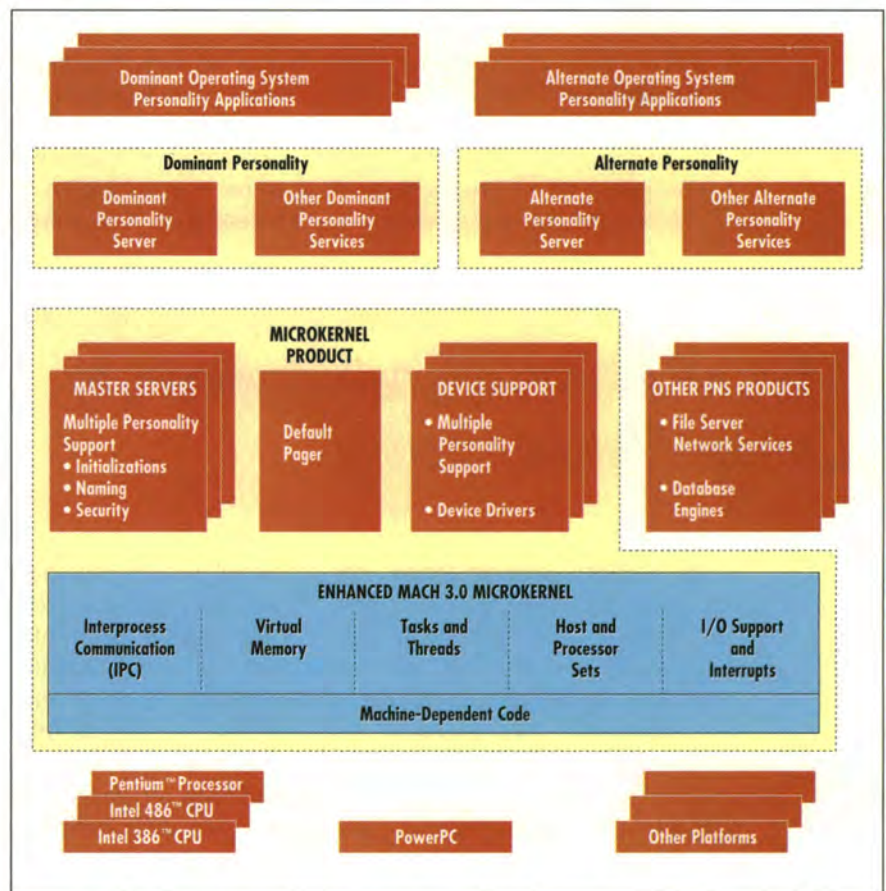


Figure 1. IBM's microkernel-based operating system structure (reprinted from IBM Microkernel Technology, IBM Corp., June 1993)

invokes the functions that correspond to each operation. For instance, data can be read from a memory object by sending a message to that object. When a task creates an object, it receives a port name, which represents the object, and port rights, which give the task restricted access rights to the object. The object concept can also be extended into a network environment, where objects can exist on any system in the network. A task can transparently



invoke these remote objects using inter-process communication.

Ports. Ports are communication channels used to send and receive messages between two tasks. The port is the basic object reference mechanism; it can be used to invoke operations on objects such as virtual memory regions, files, processors, tasks, and threads. A port is a queue managed by the microkernel, to which messages from a sender task are appended by a **Send** operation. The messages are then taken out by a recipient task through a **Receive** operation. Ports are protected by a capability mechanism so that tasks with appropriate send and receive capabilities can access them. This means that only a task with receive rights can obtain messages sent to a port and only tasks with send rights can send to it. Tasks can create ports for their own

case of blocking communication, the message is transferred directly to the receiving task when it invokes the **Receive** operation. For nonblocking communication, the message is passed in two stages, first from the sender to the kernel and then from the kernel to the recipient. IPC also supports the client/server computing environment by allowing communication between tasks running on different networks through message servers.

Virtual Memory. Microkernel virtual memory support provides large paged address spaces to the tasks. The microkernel supports the concept of memory objects when using the virtual address space. It manages the physical memory as a collection of memory objects, which can be mapped into a task's virtual address space. A memory object can be

large amounts of data between tasks. It uses a technique called copy-on-write (lazy evaluation) that allows tasks to share memory objects without copying the pages associated with those objects. Initially, the pages to be shared are marked as read only. As long as the tasks make only read reference to these pages nothing is changed. A write to any of the pages generates a page fault; the microkernel creates new copies of those pages. The microkernel also provides a mechanism to efficiently transfer large amounts of data between two tasks. The sender task sends a pointer to the data area, called "out-of-line data," to the receiving task. When the recipient receives the message, the microkernel simply maps the pages to the destination task's address space, thus avoiding the physical transfer of data between tasks.

Host and Processor Set. The host and processor set are functions used by applications that utilize advanced operating system features. The host feature gives information about the processor complex running the system, as well as functions such as date, time, and system stop and restart. The processor set features are used in multiprocessor machines to group processors into classes. These classes allow a parallel application to execute multiple threads simultaneously on different processors in the machine to achieve true parallel execution.

Server Tasks. The microkernel supports special application-level tasks called personality and personality-neutral servers that provide operating system personalities and services. There are two types of servers. Personality servers are used to create operating systems such as DOS, OS/2, and AIX, allowing multiple operating system personalities to coexist on a single machine. In contrast, traditional operating system services such as file system, device, and network services are built into personality-neutral servers. For instance, the master server may load other personality servers and they all can share a common file system

The IBM microkernel technology is based on a Mach microkernel originally developed by Carnegie Mellon University.

use; to establish communication with other tasks, they may also send port capabilities to other tasks with messages. Furthermore, using network servers, port capabilities can be extended transparently over a network to invoke objects residing on different systems. Port capabilities are highly secure because they are hard to forge or create fraudulently.

Interprocess Communication. Interprocess communication (IPC) is a mechanism that allows communication between two tasks. IPC uses messages to pass information between tasks; a task sends a message to a port and another task receives it from the port. A message consists of a fixed length header followed by a collection of data including port capabilities and references. Messages can be exchanged using blocking or non-blocking communication. In the

a virtual memory page, a set of pages, or a secondary storage object such as a file. A file object mapped into the address space can be read or written like a normal memory object. When a process terminates, its mapped file objects are automatically stored back into the file system with all changes. You can associate a port with each memory object. Ports can be used to send messages to the memory object in order to request or transmit memory object data.

The microkernel allows a server task outside the kernel to manage virtual memory. This task, known as an external pager, has access to memory objects stored on a disk. When virtual memory is allocated, an external pager may be specified to handle paging requests. If no external page is specified, a default pager is used.

The microkernel also allows tasks to share memory objects as well as transfer

Fax at a higher level



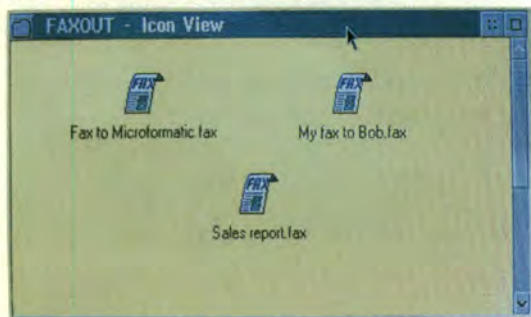
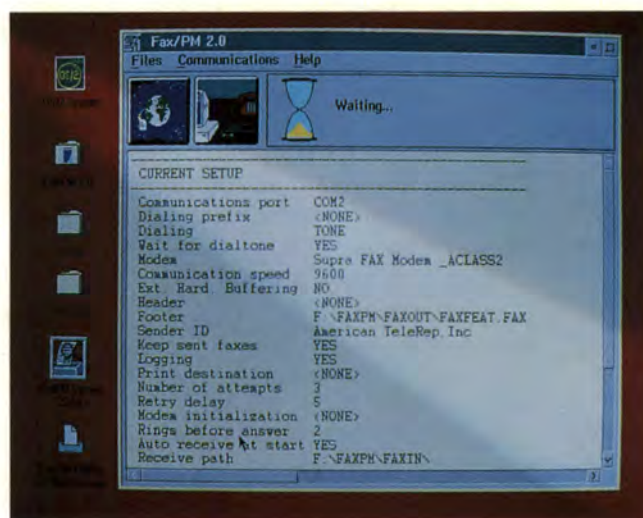
Whether you're in a Dos, Win-OS/2 or OS/2 session running under OS/2 2.1, now you have the ability to send and receive faxes - without leaving the application. Just select the Fax/PM driver as your printer. To fax...just print!

Fax/PM 32-bit also really capitalizes on the advanced features of the Workplace Shell object-oriented interface. You can send faxes right from the Workplace Shell by simply dragging and dropping a data object to the fax object.

Features like these are too good to keep to yourself, so there are also Fax/PM versions available for LAN and Client-Server environments, all CID enabled.

And applications developers will appreciate SOM APIs that allow for integration of the Fax/PM engine into custom applications. We could go on about how Fax/PM can make your PC an even more productive place.

But we'd rather let the fax speak for themselves.



with Fax/PM.

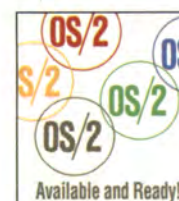
To order or to find out more about Fax/PM, call U.S.: 1-203-644-1708 / fax 1-203-648-9587
Europe: (33) 1 48 70 19 00 / fax (33) 1 48 70 27 29



Microformatic

Europe
2 rue Navoiseau
F-93100 Montreuil
France.

U.S.A.
P.O. Box 722
610 Niederwerfer Road
South Windsor, CT 06074





server, device drivers, and default pager, as illustrated in Figure 1. The servers communicate with each other through ports via interprocess communication.

MICROKERNEL HIGH-LEVEL INTERFACES

The microkernel provides two high-level program interfaces that help programmers develop multi-threaded and client/server applications.

The IBM microkernel provides the core technology for a next-generation operating system.

C Threads. The IBM microkernel provides a set of low-level functions to create and manage multithreaded programs. Using these low-level APIs gives complete control of a thread, but it also requires handling machine dependencies, requiring programmers, for example, to set up the state of a thread. The microkernel also provides a set of high-level C language interface functions called C threads that help users create multithreaded programs rather than using low-level thread functions. (These routines are contained in the C threads library.) A major benefit of the C thread functions is code portability; the library provides machine-independent functions to create and control threads. The C thread library also contains functions to coordinate multiple threads, shared variables, and mutual exclusion for critical sections. This frees the programmer from developing complex thread synchronization mechanisms.

Microkernel Interface Generator (MIG).

Programs using microkernel interprocess communication often follow the client/server programming paradigm in which clients send a message to request services from a server and receive data via another message. To relieve the programmer of the tedious task of setting up and sending messages, the microkernel provides an IPC interface generator called MIG. MIG is a compiler that takes a specification of a message-passing

interface and a procedure call interface and generates appropriate C language code for both client and server programs. This eliminates the need for the programmer to implement the packing, sending, receiving, and unpacking of IPC messages. An additional benefit is that generating IPC code through MIG eliminates the chance for programming errors caused when code is written manually.

CONCLUSION

The IBM microkernel provides the core technology needed for the development of a next-generation operating system. Its multiple operating system personality support allows applications written for different operating systems to be run concurrently on the same machine. The multiprocessor support allows applications to achieve maximum performance from multiprocessor hardware. The relatively small kernel supports real-time applications. The ability to maintain and access objects over a network allows distributed computing. Finally, the inherently secure and easily portable microkernel allows the creation of secure operating systems on a variety of hardware platforms.

Pylee Lennil, IBM Corp., 1000 N.W. 51st St., Boca Raton, Fla. 33429, (407) 443-3855. Lennil joined IBM in 1983 and is an advisory programmer in the IBM entry systems technology laboratory. He is presently involved in the development of OS/2 and spent several years in PC DOS and AIX development. He received a B.S. in physics from Kerala University in India and an M.S. in computer engineering from the University of Massachusetts at Lowell.

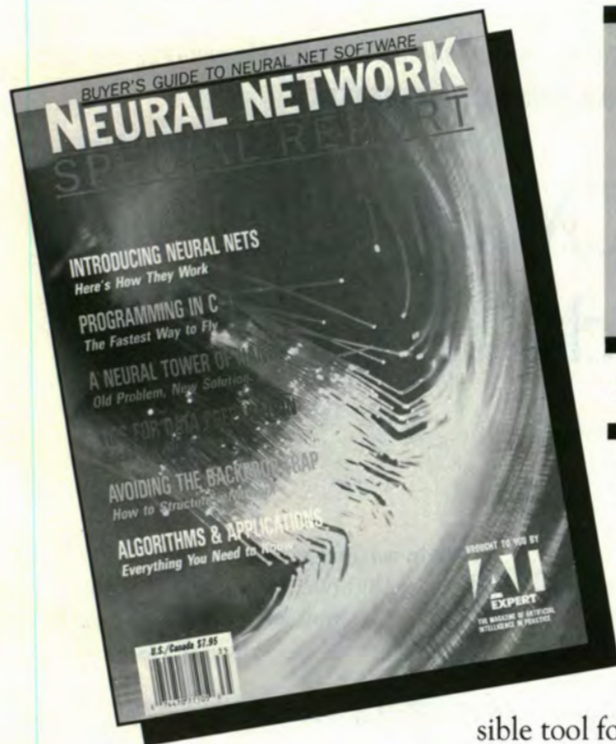
REFERENCES

- Black, David, David Golub, et. al. "Microkernel Operating System Architecture and Mach," *Journal of Information Processing*, vol. 14 no. 4, 1991.
- Boykin, Joseph, David Kirschen, Alan Langerman, and Susan Loverso, *Programming Under Mach*. New York: Addison Wesley, 1993.
- IBM Microkernel Technology, IBM Corp., June 1993



PRESENTS...

Including...
A Comprehensive Directory of
Neural Network Products!



Neural Network

SPECIAL REPORT

This publication features in-depth articles on neural networks by industry leaders such as Maureen Caudill and Jeannette Lawrence. The special report includes a comprehensive directory of neural network products.

The **Neural Network Special Report** is an indispensable tool for professionals who need to integrate neural nets into a business solution today.



GET YOUR COPY TODAY!

TO ORDER

- Telephone 1-800-444-4881
- FAX 1-415-905-2233
- Mail this order form and payment to:

AI Expert
Miller Freeman
P.O. Box 105448
Atlanta, GA 30348-5448

Send _____ copies at \$7.95 each = \$ _____

Foreign (add \$3.00 each) = \$ _____

Total Due = \$ _____

☐ Check enclosed

Charge my: ☐ VISA ☐ MasterCard ☐ AMEX

Card # _____ Exp. _____

Signature _____

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Make check payable to Miller Freeman, Inc. Orders must be prepaid in U.S. dollars.
Please allow 3-4 weeks for delivery.



While much has been written in the trade press about the Personal Computer Memory Card International Association (PCMCIA), most articles tend to focus on PCMCIA's current capabilities. This article explores the cards' potential, using an IBM PCMCIA Data/Fax Modem as an example. **BY DANA BEATTY**

Programming PCMCIA: It's All in the Cards

The PCMCIA standard has come a long way since the concept was first formulated in 1989. Initially designed only for memory card devices, it has since grown to encompass I/O implementations such as modems, 3270, Token-Ring, and Ethernet cards. Originally supported by few manufacturers, PCMCIA has matured to the point where the IBM PS/2E desktop system is dependent on PCMCIA architecture. IBM hopes that PCMCIA software will make the same progression. While other system manufacturers use PCMCIA heavily in notebook and laptop computers, IBM also uses the cards in its desktop systems.

PCMCIA cards are distinct from standard PC adapter cards in several ways:

- PCMCIA is its own bus; the cards are neither ISA bus nor Micro Channel.
- PCMCIA is nonproprietary; it is the same across IBM and non-IBM systems.
- Memory and I/O registers for the card can be mapped into any available space on the host system; users needn't deal with dip switches or system reconfiguration utilities.
- The cards can be inserted or ejected while the

system is running.

- They are lightweight and easy to use.
- Cards are completely encased; there is no need to worry about their ruggedness.

UNTAPPED SOFTWARE POWER

Because the proficiency of most PCMCIA manufacturers is limited to the actual card hardware, most of the burden for interoperability and usability of the devices has fallen on PCMCIA system and operating system providers. In the last year, IBM began shipping the key PCMCIA software elements (socket services, card services, and client drivers) with its PCMCIA systems, operating systems, and PCMCIA cards. Each of these software elements has a logical home. Figure 1 illustrates the PCMCIA software architecture.



PCMCIA card

PCMCIA Software Architecture. Socket services, the lowest software layer, interacts directly with the system hardware chip that controls the PCMCIA socket. Socket services is concerned with tasks such as fielding interrupts generated when a card is inserted or ejected from a system. Because the chip

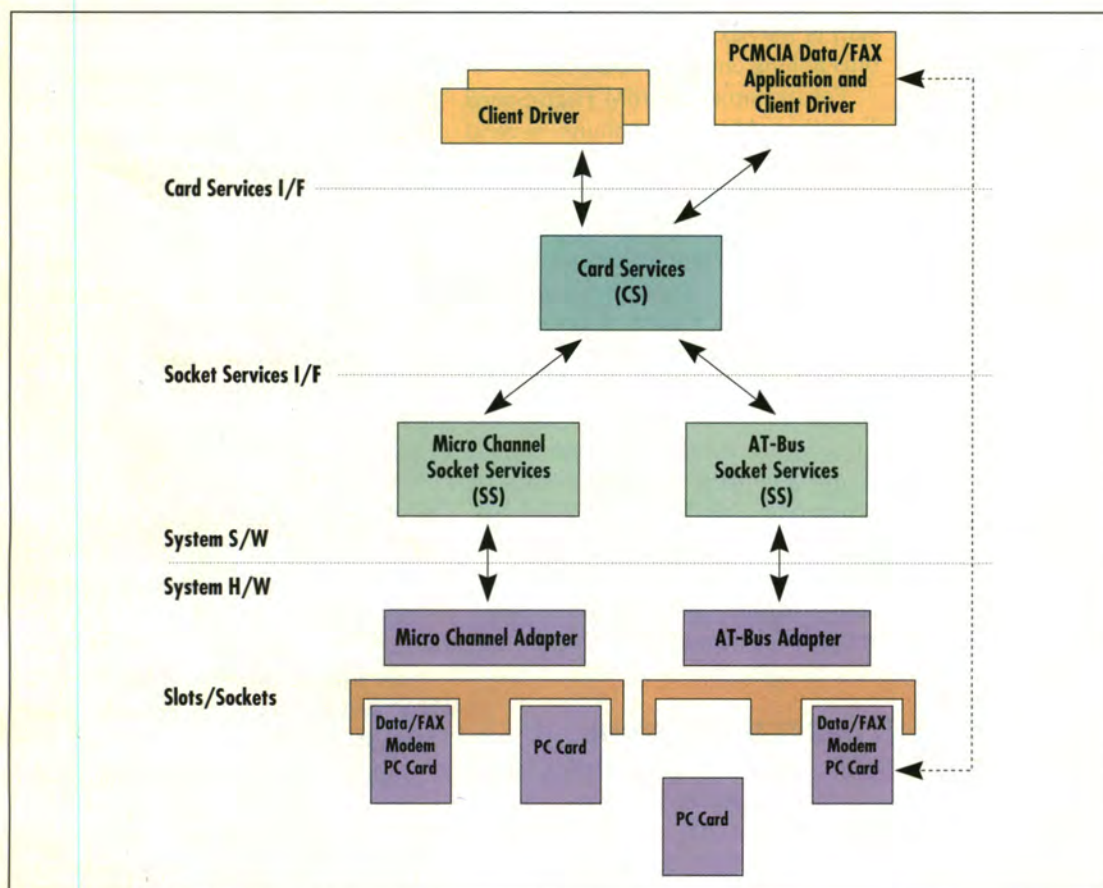


Figure 1. PCMCIA software architecture

that supports the socket varies from system to system, socket services could reside in system BIOS; today, however, it ships with the system as a device driver or similar configuration.

Card services communicate with socket services to provide a more generalized set of functions for managing PCMCIA cards. Card services' main function is to manage and coordinate the use of system resources, such as IRQs and memory space, among all PCMCIA devices in the system. It provides a set of functions accessed by client drivers through its API. Because its implementation is operating system-specific, card services would most logically be shipped with the operating system, as IBM does with DOS 6.1 and OS/2 2.1. Currently, to facilitate an interface with card services, OS/2 client drivers must be written as device drivers.

Client drivers communicate with card services to manage specific types of PCMCIA cards (such

as flash cards) or even a specific implementation of a PCMCIA card (such as the IBM PCMCIA Data/Fax Modem). As the name implies, programs register as clients of card services. During registration, a client indicates a set of events, such as card insertion and ejection, it is interested in monitoring. When an event occurs, card services notifies all clients registered for that event via the client's callback handle. For example, inserting a card into the system generates an interrupt. Socket services fields the interrupt and informs card services of the activity. In turn, card services notifies each client registered for the card insertion event. The client callback routine determines that a card has been inserted into the system. From there, the client driver interrogates the card information structure (CIS), the content and organization of which is defined by the PCMCIA standard, to determine if it is the card or type of card the client driver is managing.

Card Information Structure. The CIS is a linked list of data structures called tuples. The information contained in a tuple is used not only to initialize the card but also to automate the operations and configuration of associated applications. Figure 2 shows an

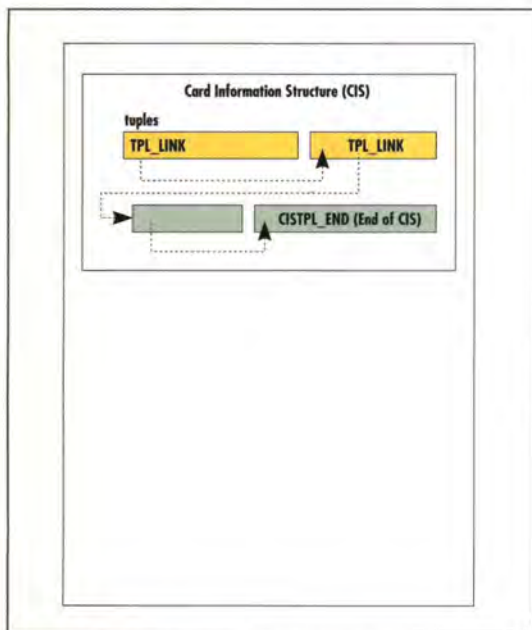


Figure 2. PCMCIA card information structure

Hex Value	Tuple	Function Description
17	CISTPL_FUNC	Function Extension Tuple
04	TPL_LINK	Link to next tuple; 4 bytes follow
00	TPLFE_TYPE	Serial description
02	TPLFE_UA	16550 UART
0C	TPLFE_UC	UART capability, even, odd parity
7F	TPLFE_UC	UART capability, 1 stop, 5-7 char

Figure 3. IBM High-Speed Data/Fax Modem serial port identification function extension tuple. The value 17x is assigned by PCMCIA for the function extension tuple (CISTPL_FUNC).

overview of the CIS organization.

A tuple cannot exceed 128 bytes in length; the first byte denotes its function. The PCMCIA specification defines

a set of tuple functions such as CISTPL_FUNC, which details functional information about the card. The second byte, known as the tuple link (TPL_LINK), contains the number of bytes until the start of the next tuple in the CIS chain, thus linking one tuple to the next.

For example, the IBM High-Speed Data/Fax Modem contains several CISTPL_FUNC tuples. Figure 3 shows the serial port identification function extension tuple, while Figure 4 shows the fax Class 2 function extension tuple. Although both of these tuples are CISTPL_FUNC types, Figure 3 details the UART capabilities while Figure 4 details the fax Class 2 capabilities of the modem. Also note that the TPL_LINK field value varies between the two tuples. Figure 3 contains only four bytes of information after its TPL_LINK field, while the tuple in Figure 4 has seven bytes. Yet both are CISTPL_FUNC tuple examples. Together, these examples show the types of capabilities described in the CIS tuples.

TYING IT ALL TOGETHER

While the power of CIS information may not be readily obvious, consider how data or fax applications are installed, configured, and operated in today's environment.

Ideally, a PCMCIA data/fax application could perform several functions. It could be a card services client driver, functioning as both an application and a data/fax client driver. When the application is invoked, it would register with card services as a client interested in the card insertion and ejection events. The application would, in this way, gain control and interrogate the card to find out whether it is a modem card and determine its capabilities.

Instead of asking for information regarding COM port assignment, speed, and even fax capabilities during installation, none of this information would be required to support a PCMCIA data/fax modem, because that information resides on the modem's CIS. The data/fax application could interrogate the CIS to make these determinations.

To do this, the client driver would

issue the card services GetConfigurationInfo() function to receive basic information (such as manufacturer identification) on the card. With this information, the client could determine if a modem were inserted.

If a modem were indeed present, the first order of business would be to allocate unclaimed resources, such as a base address and an IRQ level, to it. Because data/fax applications generally expect a modem to be mapped to architected COM port resources such as COM2 (or a base address of 02F8x and IRQ level 3), current modem client drivers resort to these same values to insure compatibility with these existing applications. This setup stifles the inherent power of PCMCIA.

But if a data/fax application were to drive the PCMCIA modem as a client driver, card services could be allowed to assign any free address space and IRQ, rather than a specific one, because the application program could adjust to accommodate the new base address and IRQ values returned by card services.

Additionally, by combining PCMCIA client driver support into the data/fax application, there is no longer a need to query end users for information such as COM port assignments during the installation process. In other words, because the application is the client driver managing the PCMCIA modem, it will request the COM port assignment dynamically when the card is inserted into the system.

WHAT'S IT ALL MEAN?

With today's modems, COM port resources are statically reserved for a device regardless of whether the device is used during that session. With PCMCIA and client drivers, resources can be acquired only when the associated device is present and any resources available when the card is inserted can be utilized. This is accomplished via the RequestIRQ() and RequestIO() functions. For example, a PCMCIA modem client driver could loop through the various IRQ levels until it successfully obtains one.

Once the data/fax application client driver acquired the necessary IRQ and I/O address space, it would then interrogate the card's CIS further. To traverse the CIS, a client can use the `GetFirstTuple()` and `GetNextTuple()` functions. These functions return the tuple function (`CISTPL_FUNCIE`) and link (`TPL_LINK`) fields, as shown in Figure 3. Alternatively, a client may advance directly to a particular tuple of interest. In the case of the PCMCIA High-Speed Data/Fax Modem, a client driver may wish to skip directly to the first `CISTPL_FUNCIE` tuple to access the modem capabilities. To do so, a client would enter the `CISTPL_FUNCIE` value into the "DesiredTuple" field within the `GetNextTuple()` function. Once card services locates the tuple, the client driver would then issue the `GetTupleData()` function to retrieve the information contained within that tuple.

In the case of the PCMCIA data/fax modem, while processing these tuples the client driver would learn the speeds supported by the card (14.4 kbps for both data and fax) as well as its fax capabilities (Class 1 and Class 2). Because it could interrogate the card, an application would not need to poll the user during installation. Additionally, should the user choose to insert a different PCMCIA data/fax modem in the future, the application would not need to be reconfigured because it would dynamically adjust itself based on the capabilities of the modem as defined on the card's CIS.

Another feature that could be incorporated into a data/fax client driver is a prompt that would alert users to insert a card in the event that they attempt to transmit a fax or data while the card is absent from the system.

Finally, when the user removes the modem from the system, socket services would field the interrupt and notify card services of the event. Card services would then notify each client driver that registered itself for the `CARD_REMOVAL` event. After determining it was the PCMCIA modem card that was ejected from the system, the modem client driver would then free the I/O address space and IRQ

resources it had allocated to that card with the `ReleaseIO()` and `ReleaseIRQ()` functions.

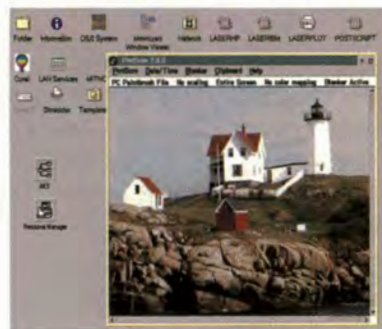
In the case of a PCMCIA modem, an existing data or fax application could be modified to incorporate the client driver functions.

THE FUTURE OF PCMCIA

The industry as a whole agrees that the future of PCMCIA is bright. The software building blocks are there, and PCMCIA seems poised for even greater success if only a "killer" application were there to exploit it.



SINGLE KEY-STROKE SCREEN PRINTING



CLIPBOARD
VIEWER

SCREEN
CAPTURE

- Quality Color or Black & White copies of color Desktop Images.
- Copy any portion of any image on the Desktop.
- Screen Saver prevents monitor burn-in.
- Date & Time Display for Time-Stamping images.
- LAN installable. Complete support for LAN attached printers.
- Economical combination of important utilities. Entity licensing available.
- The most stable and reliable product available.
- Includes both OS/2 2.1/2.0 (32-bit) and OS/2 1.3 (16-bit) versions.

THE LEADER: PrntScrn™ Screen Image Utility for OS/2



MITNOR Software
Phone: (918) 357-1628
Fax: (918) 357-2869



Hex Value	Tuple	Function Description
17	CISTPL_FUNC	Function Extension Tuple
07	TPL_LINK	Link to next tuple; here 7 bytes follow
23	TPLFE_TYPE	Fax description
03	TPLFE_UF	data rate
00		
0F	TPLFE_FM	modulation: V21, V27, V29
00	TPLFE_FY	data encryption
22	TPLFE_FS	fax feature, polling T.4
00	TPLFE_CF	country code not supported

Figure 4. IBM High-Speed Data/Fax Modem Fax class 2 function extension tuple

Dana L. Beatty, IBM Corp., 1000 NW 51st St., Boca Raton, Fla. 33431. Beatty is a staff programmer in IBM's wireless data development department. She was the lead programmer on the IBM PCMCIA Data/Fax Modem products, the first IBM

products to ship DOS and OS/2 PCMCIA client drivers. Currently the software team lead on the Cellular Digital Packet Data project, Beatty received her B.A. in computer science from the University of Texas, Austin.

REFERENCES

PCMCIA Standards (including the Card Services API) is available from PCMCIA, 1030G East Duane Ave., Sunnyvale, Calif., (408) 720-0107.

For a detailed description of the IBM PCMCIA Data/Fax Modem CIS, refer to the *IBM PCMCIA and Internal Data/Fax Modems Technical Reference* (IBM doc. S42G-2181).

The Path To A Good GUI Isn't Always Clear

Navigate the maze with the NEW 2.1 Professional Developers ToolKit

Promises are cheap, GUI development tools aren't. Gpf 2.1 demo software is **FREE**. Try Gpf before you buy any tool and you'll agree that **Gpf 2.1 DELIVERS**.

With this powerful point and click visual programming environment you can create a CUA '91 Graphical User Interface for **OS/2 Presentation Manager®** or **MS DOS/Windows®** in as little as 10% of the time required to hand code the same design.

What You See Is What You Get programming reduces weeks of coding to hours. Quickly prototype and test your interface, then let **Gpf write the code**. Gpf generates error free, structured Object Oriented C or C++ source, complete with embedded SQL for OS/2 DataBase. Custom Help and Logic are added to controls as they are drawn to create full **Client/Server** or **Stand-alone** applications!

Gpf is hosted on OS/2 2.x and generates **native code** for OS/2 2.x, OS/2 1.3.x and MS Windows 3.0 or 3.1 Of course this means maximum performance with **no run time module** and **no royalties!**

Seeing Gpf is believing!

- Design 32 or 16 bit GUIs for IBM OS/2® or Microsoft Windows®
- Be creative, **WYSIWYG** design environment
- Full application and/or **DLL** generation
- Minimal time to application delivery
- Full CUA '91 Control set, 32 or 16 bit
- **Reusable Objects**
- Automatic documentation

Gpf

Try us, **FREE** Demo Software Available
Order Gpf Pro ToolKit today for just \$1440⁰⁰
Separately: Gpf 2.1 for \$1295⁰⁰ & GpfTools for \$195⁰⁰
Or, try: Gpf 2.0 Single Platform, now available for just \$495⁰⁰

Call **Gpf Systems Inc.** at: **(800) 831-0017**

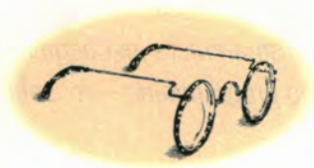


* GUI Programming Facility



SYSTEMS, INC. Phone (203) 873-3300 • fax (203) 873-3302 30 Falls Road, Moodus, CT 06469

{ figure 1. }



SPECTACLES

{ figure 2. }



OBSERVATORY

Is Your Vision In Focus?

If you're like most development managers, your vision is to build the best software possible for your company. The challenge is to focus on which new technologies offer the most immediate payback to your development objectives.

With OOP, code reusability is more of a reality. Visual tools offer shortcuts in the development cycle. Distributed architectures and GUIs offer better user access to information. Whichever of these you've decided to focus on this year, you'll find help turning your vision into reality at **Software Development '94** and **Business Software Solutions**.

Together for the first time this March, these two important industry events join forces to bring you the most extensive educational opportunity ever assembled for software development teams.



In over 200 courses you'll get hands-on help from experts with everything from programming in C++, Windows development, and GUI design, to moving your mainframe development staff to client/server, to building better information delivery systems with today's high-end tools. You'll also witness 250 vendors (like Microsoft, CA, IBM, KnowledgeWare, Borland, Gupta, Novell) race to bring you the best support for your efforts at the biggest development tools exhibition in the country.

If you're focused on building the best software for your company, take steps now to make your vision a reality by attending **SD '94 / Business Software Solutions**.

200 classes • 250 exhibitors • 25,000 of your peers



CONFERENCE & EXHIBITION

March 14-18, 1993
San Jose Convention Center

Call us at
415-905-2784
to find out more.





The Advanced Novice

In this issue we introduce a new section, "The Advanced Novice," for programmers just beginning to explore the world of OS/2. This article deals with a concept basic to object oriented programming—reusable code and reuse techniques. **BY JOSEPH H. MCINTYRE**

Reuse: Recognizing The Opportunities

For many years, the job definition of a systems analyst was identified with the term "efficiency expert" and the elimination of redundant manual processes. The work involved determining current processes, their products, and the factors surrounding and affecting, or being affected by, each process. With the collective knowledge about the processes, an analyst could create alternative methods, either by applying new models or replacing the current model with a known better model. The selected alterna-

cessful models to new processes reduces time to enact and lessens the chance of repeating shortfalls of previous implementations. The concept of reuse, of course, may also be applied to the work of the analyst.

SOME EXAMPLES

In software development today, many people consider reuse to encompass only code libraries and object-oriented programming technologies. This is, however, a narrow and restrictive view.

Knowledge is fundamentally reusable; collecting and distributing that knowledge is the challenge. Tips and Techniques documents can be collected from many users' experiences. The resulting information may be combined and released as a book or on-line reference, or placed in a repository. Users can use it to record or build detailed information suitable for their environments.

Operating system API calls are designed to be applicable to many applications, making it preferable to utilize the API instead of writing new subsystems specific to an application. For instance, in OS/2 the window management APIs are used by graphical programs. While these APIs may be extended by an individual application where desired, they would never be entirely replaced. Because PM's extensible architecture offers standard methods for implementing extensions (for example, subclassing and DLLs), working with extensions is relatively straightforward.

Test cases for an application may be run on multiple workstations with differing configura-

Knowledge is fundamentally reusable; collecting and distributing that knowledge is the challenge.

tive model could then be monitored and evaluated; if successful, it could be applied elsewhere. The efficiencies from each successful model built on previous models and became foundations for the next.

REUSE IS NOT NEW

Reuse is a cornerstone of the analyst's task. Improving processes through the elimination of redundancies can be the easiest route to identifying, assessing, quantifying cost savings, and implementing new systems. Application of suc-



tions by including setup variables or environment settings for variables such as hardware installed, network names, and connections. This allows test plans and test cases to be moved to a different environment, used in subsequent releases, or used to test companion products.

REUSE THROUGHOUT THE DEVELOPMENT CYCLE

If reuse—and the processes for developing reusable deliverables—is applied to all phases of the software development process, there are many opportunities for improvements. Each phase holds different opportunities, but deliverable candidates for reuse are best identified and prepared before they are produced. This leads to design with minimal restrictions on reuse.

In the analysis phase, reuse can eliminate repetitive work. Searches for past models, data dictionaries, new business models, and concurrent projects may yield knowledge applicable to a current project. Other projects can be enhanced by saving even the information not directly used on a project (it is also wise to retain information in case the analysis is later reviewed).

Applying reuse to architecture-level designs can yield design documents that can be applied across platforms or projects much more readily than code derived from them. It is also easier to expand or limit concepts at the design stage. The important distinction between design and code is that code is a single implementation of a design. In many cases, design documentation is much more useful to a subsequent user than is the source code itself. (At a minimum, the code can be used as a reference.) An excellent example of this is the contrast between the API reference and Programming Guides from the OS/2 toolkit. It would be very difficult to program for Presentation Manager using only the API reference without the concepts presented in the Guide.

In the coding phase, suitability for reuse is determined by a number of factors including the tools used, the physical state of the code (how modular, whether there are global variables, naming conventions, and so on), and the quality of the code (in the context of both the original application and other possible applications). The following sections give some guidelines.

Compiler Languages. It is best to consider your reuse goals before determining the language you will use to pursue them. There are several pitfalls to watch for.

If you work in a mixed-language environment (this includes a single language used on multiple platforms), it is important to anticipate translation and connection hurdles. Assembler is not easy to read or understand; libraries may be limited to a specific platform or compiler and some languages cannot call subroutines written in other languages. An emerging technology is the language-independent System Object Model (SOM), which allows the creation of bindings that let applications and objects written in different languages communicate transparently.

In many cases, design documentation is much more useful to a subsequent user than is the source code itself.

Modularity is very important. It is desirable to have the smallest meaningful component that can be utilized with minimal effort. This may refer to anything from a single function to a complete subsystem. For example, CenterWindow is a single function reusable component that positions a window in the center of another window. At the other extreme, Lotus 123 for OS/2 and Freelance Graphics for OS/2 share the same graph engine. In both of these examples, the reusable component provides a well defined, concise function to the application it serves.

If a product will be extensible by a customer, consideration of the customer is important. In some cases, source code is required, which may not be acceptable for the vendor.

Source code (not just comments) must be readable and logically organized. Well-structured code has meaningful variable names, related functions grouped logically, and well-organized modules if a task involves code in more than one source file.



Your Own Repository

This is an easy but effective way to introduce reuse concepts unobtrusively. The users learn to think about presenting materials in forms suitable for peers to make use of and see the usefulness of having a common knowledge source that is always available. (You may even want to use this concept on a standalone machine for your own work. It can be a good personal organizer, especially if you frequently 'change hats.')

Creating the repository

1. Create a directory on a network server.
2. Set up client workstations to see this directory with read, write, update, and create directory rights. Delete may be restricted.
3. Distribute simple instructions for users of the repository (this may include environment-specific information).

Using the repository

To add a new subject, create a subdirectory. Topics within a subject may be added by creating a subdirectory within a subject subdirectory.

Create a new entry, or update an existing one, with an editor/word processor. Make sure the format is compatible with other users (this should be addressed in the instructions from step 3 above).

Suggestions

Use HPFS to allow long file names to be used. Eight-letter names are not very practical or usable.

Use a navigation tool (the Drives object is an example) to move around the repository. This will save typing and file name quoting issues with long names. Double clicking on the file icon will bring up the file in the editor.

Table 1. A simple knowledge repository

Concept level from a repository object description

A phone number consists of 10 digits in the format (123) 456-7890, where (123) is an area code representing a region and the remaining seven digits represent a unique number within the region. The second digit of the area code is always 0 or 1. No area code starts with 0 or 1 and no customer number begins with 0 or 1.

Design level from a repository dictionary

Data Type: numeric, 10 digits, no decimal positions.

Validation: all digits required, if not null. First number in range 2-9 inclusive, second number in range 0-1 inclusive, fourth number in range 2-9 inclusive. Format: (123) 456-7890

Table 2. Reusable components for a phone number

Coding conventions such as module file names, file header comments, and code formats make navigation and reading easier.

Testing. In the testing phase, reuse may include maintaining test cases across

product releases, sharing test cases between products (for example, using the same services of the operating system) and bringing unit-test cases forward from programmers to testers. In both automated and manual environments, a well designed test case suite

can provide a good reuse foundation with little additional effort.

Many test cases are exactly the same except for one or two steps; the first reuse candidate is therefore the test cases themselves. Creating a test case should start with foundation cases and

be filled out with specific cases that address differences. A foundation test case may not be a full case; instead it may provide environment information and application setup on which other cases will build. Future test cases may be built by extending this foundation case, rather than using a more complicated existing case and removing unnecessary steps.

Most test cases would run identically on all machines—except for a few machine-specific variables like the network name for the workstation or a server name. Test cases should not include hard-coded names; instead, they should use aliases, variables (in the case of automated tools), or general names (for manual cases). These allow names to be configured for individual workstations.

Often, while a test case is running, a new error is found for which a test case does not exist. A new case, based on the current one, should be created to ensure that the condition will be tested. If a framework exists, generation of the new case is easy; the foundation case for the current test case can be extended to test the new condition.

Installation and Maintenance. The delivery and maintenance phases have a distinctive reuse opportunity, namely the installation and service programs. Install programs all do the same thing, more or less—copy files, set up the user environment for the application, possibly do initial program setup to customize applications for the user. Service programs check service levels and replace files; they may provide other services like rollback of changes.

There are many installation programs, ranging from simple batch programs to full graphical interface applications. Applications that come without an install program are natural candidates for reuse.

If you don't have a solid install base on which to build, products that provide this foundation can be a good starting point for creating custom

install and service programs. These include the Stirling Group's Install Shield and IBM's Software Installer for OS/2.

THE FOUNDATION FOR REUSE

The foundation for effective reuse is documentation. Specifically, the entire software cycle for a component must be available in complete, accurate, and



Rhetorex Voice Processing boards make CTI a reality.



If you're asking "what's CTI," you're missing one of the hottest new technologies going.

Computer Telephony Integration links PC-based computer applications to the telephone network, providing voice/fax mail, interactive voice response, and other telephony applications.

Interested? Maybe you're already developing a CTI application. Then it's time to discover Rhetorex.™

The fact is, when it comes to multi-port voice processing components, no one offers more value than Rhetorex.

With Rhetorex voice platforms, you'll get state-of-the-art DSP-based technology, designed from

the ground up for reliability. Sophisticated DSP algorithms give you unparalleled performance. Best of all, enhancements to algorithms are made via software upgrade.

And Rhetorex products help you develop applications quickly. All components include device drivers, a straightforward API, and sample programs. Friendly, free technical support is available at any stage of development.

For the best value in CTI technology—from our 2 and 4 port voice processing boards and fax boards, to our 24-port platform—then give Rhetorex a call. (408) 370-0881. And start making CTI a reality.



RHETOREX

Rhetorex, Inc.

200 E. Hacienda Avenue
Campbell, CA 95008-6617
Tel. (408) 370-0881
Fax (408) 370-1171

All trademarks identified by the ™ symbol are trademarks of Rhetorex, Inc.
All other trademarks belong to their respective owners. © 1993 Rhetorex, Inc.



Concept level from a repository object description

A phone number consists of 10 digits in the format (123) 456-7890, where (123) is an area code representing a region and the remaining seven digits represent a unique number within the region. No area code starts with 0 or 1 and no customer number begins with 0 or 1. For numbers assigned before 1994, the second digit of the area code is a 0 or 1. Starting in 1994, the second digit is not restricted.

Design level from a repository dictionary

Data Type: numeric, 10 digits, no decimal positions.

Validation: all digits required if not null. First number in range 2-9 inclusive; second number in range 0-1 inclusive, service year < 1994; second number not restricted, service year >= 1994; fourth number in range 2-9 inclusive.

Format: (123) 456-7890

Table 3. Changes to reflect a possible area code change scenario

sufficient documents. This allows you to determine the applicability of a component to another project with no prior knowledge of that component. Ideally (and to be effective in large systems),

grams or forms. Even paper in a central filing cabinet is a start. The important point is that there is a common repository (or repositories) that people know how to access. A simple repository

also be documented.

- Coding—Cobol copy book, C data structure, C++ class object model, and object class implementation.

Using a Dynamic Data Exchange (DDE) server in an OS/2 program.

- Concept—explanation of DDE services and where to use them; comparison to other alternatives.
- Design—implementation alternatives for creating DDE clients.
- Samples—sample implementations, possibly in pseudo-code.
- Coding—code fragments, working samples.

In both these cases, the documents are easily extensible. In fact, the phone number could apply only to North American numbers, then be extended to included worldwide numbers. Note that in the phone number example actual code implementations were provided, whereas in the DDE example samples were provided. With DDE, the subject is very dependent on its application; thus a generic solution is not likely to be useful. Templates are provided, however, as a starting point.

The important point is the availability of knowledge on a subject, not just the final implementation; not only the code can be reused; so can all the work that provided the knowledge on which the code was based. This has some

The foundation for effective reuse is documentation. Specifically, the entire software cycle for a component must be available in complete, accurate, and sufficient documents.

these documents should be in a form suitable for retrieval in an interactive browsing environment.

Absent of an electronic environment providing advanced search capabilities within the context of components, there are many other effective storage and retrieval techniques. The most basic is a collection of facts in a knowledge base, which can range in scope from a primer on a business process to an encyclopedia in a CASE tool. From this set of facts, viewers can retrieve information from the data. For a text file, many tools can provide search and hyperlink capabilities; the data in a CASE tool encyclopedia can be displayed as dia-

grams or forms. Even paper in a central filing cabinet is a start. The important point is that there is a common repository (or repositories) that people know how to access. A simple repository

A component always contains a concept; it may also contain any number of stages through one or more implementations. Following are some examples of different components.

Phone number.

- Concept—an explanation of the valid phone number sequences for all numbers around the world, include information such as the format expected by each country.
- Design—a logical data type description to implement phone number data structures. Validation information such as country codes could

Introducing
The Developer
Connection for OS/2

Development's

They're all here. All the tools, tips and techniques you need to send your OS/2® development rocketing up the charts. Brought to you by the original artists: IBM's own OS/2 development team.



Join The Developer Connection for OS/2,™ and you'll receive the most timely and extensive information available to the OS/2 community. Four times a year, you'll get a CD packed with the latest productivity tools, utilities and sample programs from IBM and others. A powerful browser and easy, graphical user interface let you locate any topic instantly in our comprehensive technical library.

Each CD includes the latest releases of smashes like "The Developer's Toolkit for OS/2," "Multimedia Presentation Manager/2 Toolkit," and "Pen for OS/2 Toolkit." Plus, you also get pre-release versions of many IBM products, operating systems, internal development tools, product demos, bit maps—you name it.

But wait, there's more. You also receive



greatest

The Developer Connection News, our newsletter filled with information about the latest OS/2 developments, new products, a Q&A column and much more. Plus, you get access to the Developer Connection forum on CompuServe,™* where you can talk directly to the experts.

And here's music to your ears: Buy a year's subscription to The Developer Connection for OS/2 before February 15, 1994 for only \$199 and receive a free copy of the Clear & Simple® hit "Performance 2.1," a tuning kit for OS/2. Call 1 800 6DEVCON today, and start producing some hits of your own.

Operate at a higher level.™

hits.



*CompuServe membership is required. IBM and OS/2 are registered trademarks and The Developer Connection for OS/2 and "Operate at a higher level" are trademarks of International Business Machines Corporation. CompuServe is a trademark of CompuServe Incorporated. Clear & Simple is a registered trademark of Clear & Simple, Inc. ©1993 IBM Corp.

obvious advantages:

- Collaborative work by experts. Each expert can provide specific knowledge, allowing those in other areas to expand the work.
- Revisions can be documented throughout the entire component, applied at the level on which the change is most appropriate, and cascaded through the other levels.
- The structure for the documentation allows different people to work within a framework.
- The structure helps people think in terms of components rather than whole solutions.

This concept seems at first to introduce a lot of work. Much of this work, however, is being done but is unrecorded. Top down, bottom up, and object-oriented design all break problems into smaller units. Reuse helps when units are discovered, during searches for an existing component that satisfies a need, or when creating a new reusable component.

Tables 2 and 3 show the before and after images of two repository documents for U.S. phone numbers. There is little work required to maintain the documents.

Information on reuse can parallel that on object-oriented techniques. Reuse, however, extends beyond object-oriented work; the concepts apply to existing processes as well. Object-oriented environments do provide additional facilities to implement reuse more effectively.

A search for an existing component can be executed in one of several ways, depending on the component and the context of the need. For instance, the phone number search could consist of a search in a CASE tool repository and information from programmers or a long distance phone company. Searches vary depending on available information and additional requirements.

Collected information should be stored in a repository (a CASE tool, a repository product, or even a text file in a known place) that makes the infor-

File: USERPROF.IDL

```
#include <somobj.idl>

interface UserProfile : SOMObject
{
    attribute string Name;
    attribute string NetworkName;
    attribute string WordProcessor;
    attribute string Spreadsheet;
    ...
};
```

File: VUSERPRO.IDL

```
#include "userprof.idl"

interface vUserProfile : UserProfile
{
    short UpdateProfile (in unsigned long hwndParent,
                        in unsigned long hwndOwner);
    ...
};
```

File: VUSERPRO.C

```
#define vUserProfile_Class_Source
#include <vuserpro.ih>

MRESULT EXPENTRY DialogProfileUpdate (HWND, unsigned, MPARAM, MPARAM);

/* UpdateProfile displays a dialog with the current user profile
   object data and allows the user to change the values. The result
   returned will be DID_OK if the user saved changes. */

SOM_Scope short SOMLINK UpdateProfile(vUserProfile somSelf,
                                       Environment *ev, unsigned long hwndParent,
                                       unsigned long hwndOwner)
{
    short sResult;

    /* Display dialog box */
    sResult = (SHORT) WinDlgBox (hwndParent, hwndOwner,
                                (PFNWP) DialogProfileUpdate, (HMODULE) 0,
                                IDD_UPDATEUSERPROFILE, somSelf);

    return (sResult);
}
```

Figure 1. Good reuse implementation (continued on page 90)

A Blue-Ribbon Panel Named **POWER TRANSLATOR™ PROFESSIONAL** Most Innovative Software of 1993...



But our users are saying things that *really* make us proud.

The prize: 1993 Discover Award
for most innovative software.

The judges: John Dvorak,
Michael Miller, Marvin
Minsky, Esther Dyson, and
Stewart Cheifet. Their Affilia-
tions: PC Magazine, PC Computing,
MacUser, Computer World, Computer Chronicles

and MIT. The winner: **POWER TRANSLATOR
PROFESSIONAL** foreign language translation
software, from Globalink®!

But our users don't think about "innovation."
They have specific work in mind.

For Eric Vogt, translator for the American
Red Cross, **POWER TRANSLATOR PROFESSIONAL**
guarantees consistency and "...saves an enormous amount
of time... A person simply can't remember 5,000 scientific
terms and how they translated each of them the last time."
For Jamala Banks, a language student at Howard Univer-
sity, "what you would normally take a day on, you can

"...it saves an enormous amount of time..."

— Eric Vogt, American Red Cross translator

"...I really expanded my vocabulary, my grammar
...using the program has been just great."

— Jamala Banks, Howard University student

now virtually do in just minutes!"

Available for DOS, OS/2®,
Macintosh® and UNIX®, **POWER
TRANSLATOR PROFESSIONAL**
features speeds up to 100 times
faster than human translators,
easily-updated dictionaries of over
250,000 words and phrases and a 90%-plus
accuracy rate! Subject dictionaries are also available, in-
cluding computer, legal, business and finance.

Find out about the software package that's got Discover
Magazine – and a lot of other people – talking. Call
Globalink today and ask about **POWER TRANSLATOR
PROFESSIONAL**, your ticket to world-wide communication.



for OS/2



lobalink, Inc.

9302 Lee Highway, Fairfax, VA 22031
Intl.: 1-703-273-5600
Fax: 1-703-273-3866

1-800-767-0035

Incredibly accurate translations: English to/from Spanish, French, or German



mation available for future use. This makes information more stable as knowledgeable people move and the knowledge they have acquired moves with them.

AN EXAMPLE OF REUSABLE CODE

Figure 1 shows a set of file fragments that demonstrate the separation of user interface code from application logic code in an object-oriented language. The example uses SOM, although the principles can be applied to other languages that support inheritance.

The application object is defined, followed by an object that inherits from it and adds the user interface functionality. The object user can decide whether to use the application object independently or with the user interface code. This setup also provides multiple user interface implementations for the single application object.

REUSE ISSUES

This section is intended to provoke further thought on the applicability and benefits of reuse techniques. Try to envision these issues in the context of your environment.

Reuse is the responsibility of the provider, not of the user. The provider contributes the knowledge and implementations to the repository. The user, on the other hand, simply selects components from the repository for integration into other products. If a component is not suitable, the user will not use it. It is therefore incumbent on the provider to provide components suitable for use, given an arbitrary and demanding user.

Components must be able to be pulled cleanly from their existing product and incorporated without disturbing the environment of the new product. Otherwise, the process becomes too difficult and leads to rework. An example of a good reuse component is an independent module that contains financial calculations for mortgages.

If a component relies on one or

```
/* DialogProfileUpdate is the dialog window callback function for
the update of the UserProfile object. */
```

```
MRESULT EXPENTRY DialogProfileUpdate (HWND hwnd, unsigned msg,
MPARAM mp1, MPARAM mp2)
```

```
{
    static vUserProfile userprofile;

    switch (msg)
    {
        case WM_INITDLG:
            /* get object passed in */
            userprofile = (vUserProfile) mp2;
            ...
    }
    ...
}
```

File: MAIN.C

```
...
vUserProfile userprofile;
Environment *ev;

/* create new object */
ev = somGetGlobalEnvironment ();
userprofile = vUserProfileNew ();

/* set values for object attributes */
...

/* display update dialog */
_UpdateProfile (userprofile, ev, HWND_DESKTOP, HWND_DESKTOP);

/* release object */
_somFree (userprofile);
...
```

Figure 1. Good reuse implementation (continued from page 88)

more other components, the interfaces should be clearly defined such that the dependencies can be replaced with functional equivalents. Dependencies should be documented so the dependent components' interactions with their dependencies need not be known.

Documentation should be complete, accurate, and sufficient. The benefit of

reuse is not realized if the user must do as much work using the component as writing a new one. All documentation should be maintained with the component; changes cannot be made at one level without updating the appropriate information in the other levels. The update in other levels may implement the changes or indicate the level to which the component complies if it

does not fully implement the changes. For instance, the design document for phone numbers could include country code, but the Cobol copybook may not implement international numbers. This would be stated in the Cobol copybook documentation.

Reuse requires thought. It also requires vision to recognize some less obvious opportunities for reuse. Reuse can also inspire better solutions, adding a context in which picturing a component acts as a catalyst for new ideas.

Applying reuse concepts does not require a new set of tools; these concepts can be applied to existing frameworks and extended to fit the requirements of your environment. Practically, enabling tools are necessary to facilitate searches through a large number of components and provide repository services. But even if the concepts are applied only to individual work, they can provide insight into the function and benefit of the techniques.

REFERENCES

Booch, Grady. *Object-Oriented Design with Applications*. Redwood City, Calif.: Benjamin/Cummings Publishing, 1991. (ISBN # 0-8053-0091-0)

SOMobjects Developers Toolkit (IBM Doc. 96F8647)

Install Shield, The Stirling Group

Software Installer for OS/2 (IBM Doc. 5621-434)

CONCLUSION

Reuse is often taken for granted. A goal of many development methodologies, it is being incorporated in a number of ways. Understanding the concepts and deliverables of a system in which reuse is utilized provides a clear vision of the environment and possible benefits.

Joseph H. McIntyre, *Analysts International Corp. (AiC), 9 Village Circle Suite 110, Roanoke, Texas 76262*. Before his current position working on OS/2-based database tools for IBM, McIntyre worked on a number of corporate client/server and cooperative processing development projects. He holds a degree in business information systems from Fanshawe College, London, Ontario, Canada.

WATER

Water is not the pure and natural resource it once was. The effects of global warming, acid rain, and chemical technology have tampered with it. To protect our water from pollution, we ask that you think about how you dispose of chemicals, garbage, and other contaminants.

Here are some helpful hints.

Adhering to these tips will benefit the water you drink, the water you swim and bathe in, and the food you eat.

Another set of helpful hints from the Green-Keeper Series, sponsored by the MFI Green Project of Miller Freeman, Inc., San Francisco, CA.

- 1. Keep Your Trash Out of the Gutter:** Sweep your driveways and sidewalks instead of hosing them with water. Put trash in the can instead of the gutter.
- 2. Get Rid of Garden Pests:** Put ladybugs or praying mantis in your garden instead of using insecticides. Contact your local nursery for alternative ideas to herbicides and pesticides.
- 3. Fix Your Car Leaks:** Recycle your motor oil at your local mechanic or garage. Don't dump it into a storm drain. More oil enters beaches and lakes from urban runoff than from tanker spills.
- 4. Use Non-Toxics:** Buy earth-friendly products now. Vinegar and baking soda are excellent cleaning agents.



This article presents a programmer's view of LAN NetView and describes how to develop an application with the X/open XMP/XOM functions. In many cases, XOM functions are used to help build the objects and convert between private (used by the framework), and public (used by the developer) structures. BY JACOB M. RZEPKA and CHUCK R. MCKELLEY

Introduction To LAN NetView Application Development: An Xmp/Xom Primer



Jacob M. Rzepka



Chuck R. McKelley

This article introduces the concepts of XMP and XOM, which represent the primary programming interface for LAN NetView. It covers building and analyzing the data structures associated with XMP/XOM and constructing simple management requests to agents developed for LAN NetView. In an upcoming article, we will unite these programming concepts with the actual agent implementations and framework facilities provided by the LAN NetView Manager/Enabler products.

As described in the first article of this series ("LAN NetView: A Programming Overview," *OS/2 Developer*, July/August 1993), LAN NetView is a platform or framework for the development and execution of applications used to manage network resources. It should not be confused with network managers, as it can manage a system rather than just a network. "System" here refers to applications and physical resources that can be addressed with Agents supplied or written by various resource owners. Examples of resources include IBM's Database Manager, Communications Manager/2, OS/2 2.0 and 2.1 products, router hubs, adapter cards, and so on. As an analogy, LAN NetView can be considered an operating system for management applications. It performs or provides many of the services common to all management applications, including memory management, transport services, request dispatching, and information routing.

There are currently three major protocols that define management interfaces, Simple Network

Management Protocol (SNMP), Common Management Interface Protocol (CMIP), and SNMP 2. While the relative merits of each protocol are beyond the scope of this article; the subject has been covered extensively. At a high level, CMIP provides much more robust functionality than SNMP, but not without significant overhead in the implementing software and increased complexity in the specification of the operations to be performed. Today, although CMIP is gaining popularity, the vast majority of applications are written to the SNMP specification, and LAN NetView provides support for both. Where not otherwise indicated, this article focuses on programming to the CMIP specifications.

A final topic before getting into coding specifics is the transport of data between managing and managed nodes. By its very nature, SNMP uses TCP/IP to flow information between nodes. CMIP, unhampered by this restriction, has both CMOT (CMIP over TCP/IP) and CMOL (CMIP over Logical Link Control, or LLC). Within the LAN NetView framework, it is not vital that a developer understand the underlying transport to develop effective applications. All software written to function using the CMIP structures will operate transparently in both the CMOL and CMOT environments. LAN NetView's supporting APIs are not concerned primarily with the transport mechanism. APIs for SNMP and CMIP are similar and, as mentioned above, little or no difference exists between CMOL and CMOT with respect to application development.



PROGRAM DESIGN CHARACTERISTICS

The LAN NetView products implement the X/Open Management Protocol (XMP) and X/Open Abstract Data Manipulation (XOM) API specifications and its implementation adheres closely to the X/Open specifications. Although there are a few minor differences, they typically have little or no impact on the developer, representing a proper subset of the allowed function defined by the X/Open specifications. This feature is important since it assures some measure of portability with other platforms claiming support for XMP and XOM APIs.

At a block diagram level, programs using the XMP/XOM API calls have a similar structure. The program is divided into four segments: object or data definition, initialization, operational, and termination.

Addressing application development, we begin with the initialization and operational stages of defining the application. These, in turn, require the use of data structures that form the core of the application's definition segment. This article is intended as an introduction to building the required data structures, issuing management requests, and interpreting the results. The *LAN NetView Developer's Reference Manual* and *LAN NetView Administrator's Guide* provide more comprehensive detail on the framework and programming structures. A set of samples are also provided with the product.

XMP

The XMP/XOM API set is shown in Figure 1, which consists of collections of management verbs, administrative functions, and data manipulation functions. The protocol functions of XMP are broken down to request functions and the corresponding response functions. The ability to mandate a confirmation or response is a configurable characteristic of the XMP API. These functions consist of the prefix `mp_`, the function name, and a suffix of either `_req` or `_rsp`. For example, the GET function is coded as `mp_get_req()` and the response `mp_get_rsp()`.

The administrative functions, which also have a `mp_` prefix, are used to initialize or terminate an application within the NetView platform. All data constants and macros are written in capitals, while

all functions are lowercase. In the following examples, the macros and functions that contain the prefixes `om_` or `OM_` are XOM functions macros and data constants used for the definition, manipulation, or deletion of data within defined data structures.

The general structure of a LAN NetView application (not including the data definition aspects) is shown in Figure 2. At a minimum, all LAN NetView programs must contain `mp_initialize`, `mp_version`, and `mp_bind` function calls. These three functions establish a connection to the platform from within an application. The function `mp_initialize` is called first to define a workspace or data area, which holds the data structures that XMP and XOM require to operate properly. The second function, `mp_version`, defines characteristics of the interface and provides for loading of common routines used to process data within the application. Finally, `mp_bind` establishes the pathway to the framework that generates and returns responses to management requests.

All LAN NetView programs must contain `mp_initialize`, `mp_version`, and `mp_bind` function calls, which establish a connection to the platform from within an application.

Since OS/2 is a multithreaded environment, it is possible to have multiple `mp_bind` functions for a single initialization request. This function is useful when management requests are responded to infrequently. For example, a request may report when a security or access violation occurs. Since these events are rare, a separate thread can be spawned for each resource for which security must be tracked, with each thread accomplishing totally different management functions and having its own `mp_bind` and specific management request. Although there is a limit of 30 `mp_bind` functions per `mp_initialize`, multiple workspaces



CMIS Service	SNMP Service	XMP Functions	Descriptions (of request only)
Action	--	mp_action_req() mp_action_rsp()	Requests that the responder perform one of the actions defined for an object
Cancel Get	--	mp_cancel_get_req() mp_cancel_get_rsp()	Requests that the responder terminate servicing an earlier asynchronous "get" request that has not yet completed
Create	--	mp_create_req() mp_create_rsp()	Requests that the responder create an instance (object) of the specified object class
Delete	--	mp_delete_req() mp_delete_rsp()	Requests that the responder destroy a particular instance (object) of an object class
Get	Get	mp_get_req() mp_get_rsp()	Requests that the responder supply the value(s) of one or more object attributes
Set	Set	mp_set_req() mp_set_rsp()	Requests that the responder modify the value(s) of one or more object attributes
Event Report	Trap	mp_event_report_req() mp_event_report_rsp()	Issues one of the notifications (events or traps) defined for an object
--	Get Next	mp_get_next_req() mp_get_next_rsp()	Requests that the responder supply the type (name) of the next SNMP variable in the object
XOM Functions			
Function Name	Description		
om_copy()	Creates an independent duplicate of a private OM object		
om_copy_value()	Copies a string from one private OM object to another		
om_create()	Creates a new private OM object of a particular class		
om_decode()	Creates a private OM object that represents an encoded private OM object (see om_encode() below)		
om_delete()	Deletes a private or service-generated OM object		
om_encode()	Creates a new private OM object that encodes an existing private OM object using the Basic Encoding rules for ASN 1		
om_get()	Creates a public copy of all or part of a private OM object		
om_instance()	Tests an OM object for membership in a particular OM class		
om_put()	Puts attribute values of an OM object (public or private) into a private OM object		
om_read()	Reads a segment of a string in a private OM object		
om_remove()	Removes and discards an attribute value from a private OM object, or from the entire attribute		
om_write()	Writes a segment of a string into a private OM object		

Figure 1. XMP functions to support CMIS and SNMP services

can be defined with multiple calls to the `mp_initialize` function. This application is diagrammed in Figure 3.

The previous discussion centers on the fact that XMP/XOM uses synchronous calls as its default mode. With synchronous calls, a thread or process blocks or waits until the function execution completes; whereas if an access violation never occurs, each thread waiting for an occurrence is blocked indefinitely. The platform also allows for asynchronous operation, in which control continues when a request is made. When the operation completes, the requesting application is signalled and the resulting data can be processed. In the previous example, a single thread or process using a single `mp_bind` could issue multiple requests asynchronously for access violation data, as shown in Figure 3.

Once a bind is completed, it is possible to issue a management request. For CMIP, possible requests include `mp_get_req`, `mp_set_req`, `mp_action_req`, `mp_create_req`, `mp_event_report_req`, `mp_delete_req`, or `mp_cancel_get_req`. Each function has a similar parameter set, hence the notion that these represent a single API of the form `mp_xxx_req()` or `mp_xxx_rsp()` (see Figure 4). This article focuses on the `mp_set_req` function.

For SNMP, only four functions are used: `mp_get_req`, `mp_set_req`, `mp_get_next_req`, and `mp_event_report_req`. The `mp_get` will be described in a later article; its relationship to the other functions can be inferred with little difficulty.

PARAMETERS

At this point we are ready to write our first application segment, shown in Figure 5. Notice that the result of each API call is a status; this is important because only when the status indicates success is a valid result guaranteed. The result of each API call must be checked for a successful return before proceeding to the next function call.

While it shows structure, Figure 5 does not identify the parameters associated with each call. The definition of the parameters leads to the data defini-

```
workspace=mp_initialize ( );
feature_list [0] = MP_CMIS_PACKAGE;
status=mp_version (workspace, feature_list));
if (status != MP_STATUS) {
    printf {" mp_version error" \n);
    (void) om_delete (status);
    return (-1);
}

status=mp_bind(MP_DEFAULT_SESSION, workspace, &session);
if (status != MP_STATUS) {
    printf {" mp_bind error" \n);
    (void) om_delete (status);
    return (-1);
}

status=mp_set_req(MP_DEFAULT_SESSION, MP_DEFAULT_CONTEXT,
                  argument, &result, &invoke_id);

:

mp_unbind (session)
om_delete (session)
mp_shutdown (workspace)
```

Figure 2. Simplified structure of LAN NetView application

We'd like to say a word about client/server development.

TESTING.

You've chosen the platform, architecture, development tools, and application. Now's the time to decide how you're going to test your client/server applications.

The Softbridge Automated Test Facility (ATF) lets you fully automate regression testing, letting you test "often and early" in the development cycle. And that means lower costs, decreased time to release, and higher quality results.

If you're building client/server (or stand-alone) applications under OS/2, Windows, or NT, you should be using ATF to test them. To learn more about ATF, call 800-955-9190.

 Softbridge, Inc.
617-576-2257 (Phone)
617-864-7747 (FAX)

ATF: The final word in testing.





tion aspects of programming to the framework and into the utilization of XOM functions (used in manipulation and creation of the complex data structures used by the LAN NetView platform). As most developers discover, while the operational coding is very simple, the definition of parameters in the operational statements and the results of a management request tend to be rather complex.

OBJECTS

XMP and XOM rely on a definitional notation that specifically and unambiguously defines a piece of information. Within the X/Open specifications, these are referred to as objects. Objects are arrays of triplets called descriptors. The first descriptor in the array is a header and the last consists of nulls, as in Figure 6. The standard programmatic representation is shown in Figure 7.

Each descriptor consists of a type syntax and a value. The **type** field identifies the descriptor, while **syntax** identifies the type of data represented by the descriptor and **value** contains either the data or a pointer to the data. For example, a descriptor may depict:

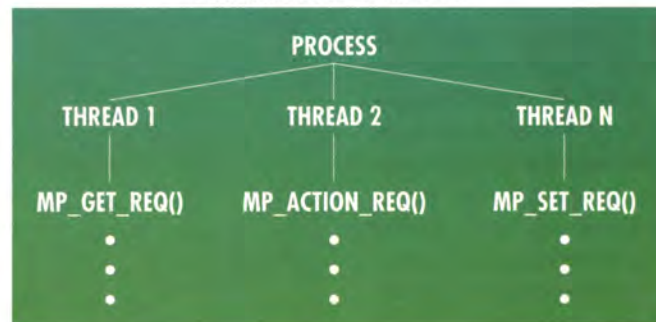
```
memory_size, integer, 300
```

Here the type is **memory_size**, its syntax is **integer**, and the value is 300. Descriptors can be more complex; the syntax field can describe a string or an object type (a pointer to an array of descriptors). For example, memory size can be represented in bytes, pages, or kilobytes. The descriptor definition might then have the structure shown in Figure 8. Structures can become very complex, with multiple levels of nested object definitions.

PACKAGES

To begin coding, a developer must define the "packages" used by the application. Each package is a collection of data structures defined for use with an associated agent or agents and represented by all or part of a DLL.

SYNCHRONOUS OPERATION



ASYNCHRONOUS OPERATION



Figure 3. Multiple workspaces defined with multiple calls to the *mp_initialize* function

C-API Function (XMP API)

```
MP_status ms_xxx_req(
    OM_private_object session,
    OM_private_object context,
    OM_object argument,
    OM_private_object *result_return
    OM_sint *invoke_id_return);
```

MP_status	Success or an error object
Session	Management service characteristic
Context	Call specific characteristic
Argument	Function and View Specific Object
Result	Return-Function Dependent
Invoke-Id-Return	ID for Asynchronous Calls

Figure 4. CM-API function (XMP API)

Packages are identified in an object of type **mp_feature**, which is usually statically defined. A sample feature list definition is given in Figure 9, section G. The feature list is used in the **mp_version** statement; the result of the **mp_version**

function call can be used to check which DLLs are loaded. Unless space is at a premium, it is easiest to include all available packages in the feature list definition and reuse the code in each application.


```

ws=mp_initialize();      /* create a work area          */
status=mp_version(...); /* define the interface          */
status=mp_bind(...);    /* Establish Connection to framework */

status=mp_set_req(...);
status=mp_unbind();      /* sever connection to framework */
status=mp_shutdown();    /* relinquish space                */

```

Figure 5. Application segment structure

PROGRAM INITIALIZATION

The `mp_initialize`, `mp_version`, and `mp_bind` functions are used to establish a connection to the framework. `mp_initialize` returns a handle to a data area used by successive functions, while `mp_version` identifies the packages used by the application. The general form of the function call is

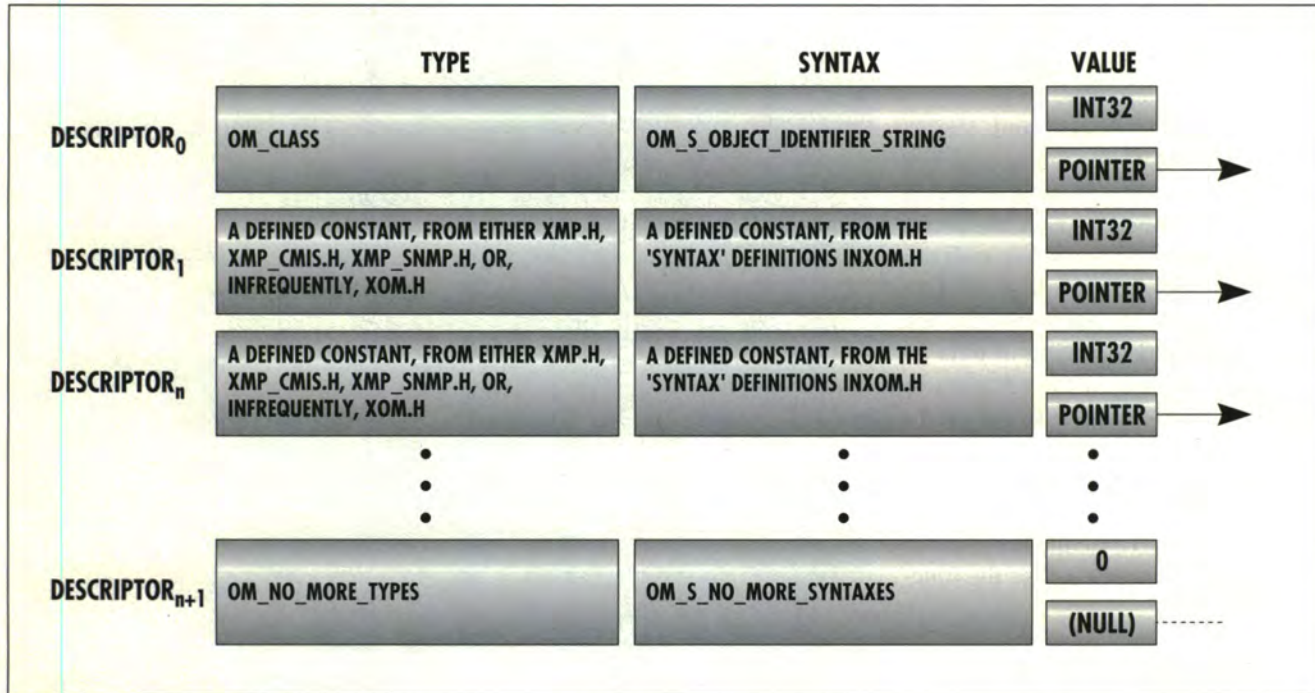


Figure 6. Array structure

```

static OM_descriptor name = {
    OM_OID_DESC(OM_CLASS, CLASS_NAME), /* Header Macro          */
    {type,syntax,value},
    {type,syntax,value},
    OM_NULL_DESCRIPTOR };              /* macro representing nulls */

```

Figure 7. Standard programatic representation of array structure

```

memory_size, object, memory_struc_ptr

memory_struc_ptr --->

header
memory_in_bytes, integer, 0
memory_in_pages, integer, 0
memory_in_Kb, integer, 0
null,null,null

```

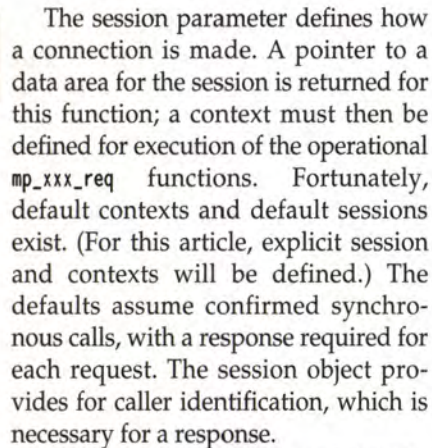
Figure 8. Descriptor with syntax field describing memory size

```
mp_version(feature_list,ws);
```

As noted earlier, `feature_list` describes the packages to be loaded for application processing. Packages are collections of definitional structures that describe the data represented in the application. They are necessary to handle differences between platforms for representation of string and numeric information. For simplicity, use the feature list definition given in Figure 9, section G. The `ws` parameter is returned from the `mp_initialize()` function.

The final initialization function is `mp_bind`. Its syntax is

```
mp_bind(session,ws,&bound_session);
```

To build the `CONTEXT` and `SESSION` objects, two XOM functions, `om_create` and `om_put`, must be introduced. (Note that all XOM functions have the prefix `om_`. XOM functions deal solely with the definition and modification of data structures, whereas XMP provides for the creation and reporting of data or operations, based on the contents of data structures.) The first step is to build a shell of the structure of interest using `om_create`, which allows the definition of an object of the class specified as a parameter. To define an instance of the `CONTEXT` object, exercise the function call:

```
_create(MP_C_CONTEXT, OM_TRUE, ws, &con-
text);
```

This builds an empty context object pointed to by `&context`. The second function, `om_put`, allows us to complete the fields in the already-created context object. Its syntax is:

```
om_put(context, OM_REPLACE_ALL, con-
text_opts, 0, 0, 0);
```

`om_put` copies descriptors from the “public” object `context_opts` into the “private” context object. These public-to-private conversions must be performed on many objects used in the framework. Public versions of the objects can be modified directly, which is why they are created; however, private versions of the objects are most efficiently used by the platform. The variable `context_opts` can be defined as an object

```

static OM_descriptor context_opts   ={\
    OM_OID_DESC(OM_CLASS, MP_C_SESSION),\
    {MP_MODE, OM_S_ENUMERATION,MP_T_CONFIRMED},\
    {MP_ASYNCHRONOUS,OM_S_BOOLEAN,OM_FALSE}\
    OM_NULL_DESCRIPTOR };\

static OM_descriptor req_title   ={\
    OM_OID_DESC(OM_CLASS, MP_C_ENTITY_NAME),\
    {MP_ENTITY, OM_S_PRINTABLE_STRING,OM_STRING("TEST")},\
    OM_NULL_DESCRIPTOR };\

static OM_descriptor session_opts   ={\
    OM_OID_DESC(OM_CLASS, MP_C_SESSION),\
    {MP_REQUESTOR_TITLE, OM_S_OBJECT,{0,req_title}},\
    OM_NULL_DESCRIPTOR };\

static OM_descriptor bmoc   ={\
    OM_OID_DESC(OM_CLASS, MP_C_OBJECT_CLASS),\
    {MP_GLOBAL_FORM,OM_S_OBJECT_IDENTIFIER_STRING,\
    {5,"\x2b\x6\xC6\x2a\x01"}},\
    OM_NULL_DESCRIPTOR };\

static OM_descriptor pub_network_id   = {\
    OM_OID_DESC( OM_CLASS, C_GNM_NAME_TYPE),\
    {GNM_P_STRING, OM_S_GRAPHIC_STRING, OM_STRING( "n" )},\
    OM_NULL_DESCRIPTOR };\

static OM_descriptor pub_netid_ava   = {\
    OM_OID_DESC( OM_CLASS, MP_C_AVA ),\
    {MP_NAMING_ATTRIBUTE_ID, OM_S_OBJECT_IDENTIFIER_STRING,\
    OM_STRING(OMP_O_GNM_A_NETWORK_ID )},\
    {MP_NAMING_ATTRIBUTE_VALUE, OM_S_OBJECT,\
    {0,pub_network_id}},\
    OM_NULL_DESCRIPTOR }; static OM_descriptor\
ds_rdn   ={\
    OM_OID_DESC(OM_CLASS, MP_C_DS_RDN),\
    {MP_DISTINGUISHED_NAME, OM_S_OBJECT,{0,ava}},\
    OM_NULL_DESCRIPTOR };\

static OM_descriptor ds_dn   ={\
    OM_OID_DESC(OM_CLASS, MP_C_DS_DN),\
    {MP_DISTINGUISHED_NAME, OM_S_OBJECT,{0,ds_rdn}},\
    OM_NULL_DESCRIPTOR };\

```

Figure 9. Sample feature list definition (continued on page 99)



that contains context variable definitions to be changed from the default state. Similarly, the session object must be created using `om_create` and `om_put` functions.

The next function is `mp_set_req`. Its general syntax is:

```
mp_set_req(bound_session, context,  
argument, &result, &invoke_id);
```

In the parameter list, the last two parameters define the result. For a synchronous case (defined in the context object), the answer or result of the function call is returned in the result variable. For an asynchronous case, a tag or identifying integer is returned in the `invoke_id` and the result parameter is meaningless. (It is the developer's responsibility to keep track of the `invoke_ids` after each asynchronous function call; this will be covered in a later article on programming for the LAN Netview platform.) The first two parameters, `bound_session` and `context`, are obtained from the `mp_bind` and the `om_put` respectively. The last argument parameter is a complex data structure describing the data to be modified.

The argument form is described as a CMIS Set Argument with the following definition, obtained from the X/Open XMP specification for the CMIS Set Argument. (Only the required descriptors are shown):

```
base managed object class  
base managed object instance  
modification_list
```

Each descriptor has a syntax of object, which implies some level of nesting. While it is possible to construct the argument dynamically, it is given in Figure 9, Section F, as a static "public" definition, which will then be copied into a private definition with `om_create` and `om_put`.

We are now faced with defining the objects `bmoc`, `bmoi`, and `mod_list`. The lowercase text indicates developer-chosen names.

Notice that the definition of base

```
static OM_descriptor bmoi ={\n    OM_OID_DESC(OM_CLASS, MP_C_OBJECT_INSTANCE),\n    /* Section D*/\n    {MP_DISTINGUISHED_NAME, OM_S_OBJECT, {{0, ds_dn}}},\n    OM_NULL_DESCRIPTOR };\n\nstatic OM_descriptor attr_id ={\n    OM_OID_DESC(OM_CLASS, MP_C_ATTRIBUTE_ID),\n    {\n    MP_GLOBAL_FORM, OBJECT_IDENTIFIER_STRING, {MP_BUFFERS}},\n    OM_NULL_DESCRIPTOR };\n\nstatic OM_descriptor mod_list ={\n    /* Section E*/\n    OM_OID_DESC(OM_CLASS, MP_C_MODIFICATION),\n    {\n    MP_ATTRIBUTE_ID, OM_S_OBJECT, {{0, attr_id}}},\n    MP_ATTRIBUTE_VALUE, OM_S_INTEGER, 10},\n    {\n    MP_MODIFY_OPERATOR_VALUE, OM_S_ENUMERATION, {{MP_REPLACE}}}\n    OM_NULL_DESCRIPTOR };\n\nstatic OM_descriptor argument ={\n    OM_OID_DESC(OM_CLASS, MP_C_CMIS_SET_ARGUMENT),\n    {\n    MP_BASE_MANAGED_OBJECT_CLASS, OM_S_OBJECT, {{0, bmoc}}},\n    /* Section F*/\n    MP_BASE_MANAGED_OBJECT_INSTANCE, OM_S_OBJECT, {{0, bmoi}}},\n    MP_MODIFICATION_LIST, OM_S_OBJECT, {{0, mod_list}}}\n    OM_NULL_DESCRIPTOR };\n\nMP_feature feature_list = {\n    OM_STRING(OMP_O_OM_OM), OM_OM_TRUE,\n    OM_STRING(OMP_O_OM_MP_COMMON_PKG), OM_TRUE,\n    /* Section G*/\n    OM_STRING(OMP_O_MP_CMIS_PKG) < OM_TRUE,\n    {{0, NULL}}, OM_FALSE};\n\nws=mp_initialize();\nstatus=mp_version(feature_list, ws);\nom_create(MP_C_CONTEXT, OM_TRUE, ws, &context);\nom_put(context, OM_REPLACE_ALL, context_opts, 0, 0, 0);\nom_create(MP_C_SESSION, OM_TRUE, ws, &session);\nom_put(session, OM_REPLACE_ALL, session_opts, 0, 0, 0);\nmp_bind(session, ws, &bound_session);\n/* Section G*/\nom_create(MP_C_CMIS_SET_ARGUMENT, OM_TRUE, ws, &arg1);\nom_put(arg1, OM_REPLACE_ALL, argument, 0, 0, 0);\nmp_set_req(bound_session, context, arg1,\n    &result, &invoke_id);\nstatus=mp_unbind(&bound_session);\nstatus=mp_shutdown();
```

Figure 9. Sample feature list definition (continued from page 98)



managed object class (bmoc object) is complete, in that a data value of type `string` has been assigned. (Within a real application a "real" class identifier would have to be selected. These classes, defined in the *LAN NetView Managed Object Catalogues*, describe classes such as the operating system and the machine.) We can now proceed to the nested object `bmoc`. Because of the nesting, it is necessary to define the objects `ds_dn`, `ds_rdn`, `AUA`, and `network_id`. This procedure is shown in Figure 9, Sections C and D.

The definition of the base managed object instance is now complete. The data used in the instance descriptor is obtained from the specification of an

specifications its structure is defined as:

```
attribute_id
attribute_value
modify_operator_value\
```

The only object not completely described is the attribute id for the item to be modified. (It is important to note that all attributes are represented by an attribute id. For example, the `buffer` variable is actually identified by an attribute id of the form 1.3.66.77.62.2.)

At this point, we have described the definition of a public object for a set argument; the code definition can be reused for all applications that need a set argument. Data portions can be modified using simple C notation. For

issue the set request. The complete code appears in the CompuServe OS2DF2 forum, library 13, under the name `XMPDEMO.C`.

Creating the private version of the set argument is optional. Doing so, however, can simplify memory management and improve performance.

At this point, all coding necessary for the execution of `set` has been completed. A complete code segment can be found in `XMPDEMO.C`, shipped as a sample with the LAN NetView Managed object product.

HOUSEKEEPING

One of the more troublesome portions of the framework, and of the X/open API in general, is that the function calls allocate space and return pointers to that space. This is true for results returned from the `mp_xxx_yyy()` API and the `om_create` and `om_get` function calls from XOM. Failing to delete structures no longer in use can lead to uncontrolled growth in the memory necessary to run an application and eventually cause a `swapper.dat full` error. It is easy to delete unnecessary structures with the `om_delete()` function:

```
om_delete(result);
om_delete(arg);
```

The only caveat is that deleting a nonexistent entity can cause an abnormal exit from the program. On the other hand, if unused data is not deleted, an application will require more and more space to continue execution. As a rule, make sure that all public and private objects are deleted once they are no longer in use. Because each new call to an XOM/XMP function results in a new area being allocated, there is no advantage to keeping unused structures around.

In general, an argument structure must be defined only once. Once you have a set of these for each `mp_xxx_req` function, they can be easily reused.

instance of an object at LAN NetView installation. (Only a portion of the instance is given in the previous figures; a complete version can be found in the CompuServe OS2DF2 forum, library 13, under the name `XMPDEMO.C`.) Our next article will give a more thorough description of the relationship between object instances and LAN NetView. The actual structural specification of the `objects` and `sub_objects` is found in the X/Open book under "base managed object instance." Everywhere an object is specified as a syntax, an additional level of definition must be accomplished. Notice the relationship in the class name header and structure definition. For example, an `AVA` object's class is `MP_C_AVA`. (The descriptors comprising the class are again found in the X/Open book under the class name.)

The last structure to be defined is the modification list. In the X/Open

example, to dynamically change the attribute value to 30, the C code `mod_list 2 .value.integer=30;` could be used.

In general, an argument structure must be defined only once. Once you have a set of these for each `mp_xxx_req` function, they can be easily reused.

At this point, we can show the structures associated with the `CONTEXT` and `SESSION` objects. Recall that we needed to define some data so the `om_put` could place these values in the objects.

For `SESSION`, we need to include a title so a response can be routed back when a request is made. For context, we will specify confirmed mode and request that the result be returned synchronously. These can be seen as the `req_title`, `session_opts`, and `context_opts` definitions in Figure 9, Section A.

Once the static structures are defined the code is relatively complete; it is time to build the argument and

Charles R. McKelley, Jr., IBM Corp., 11400 Burnet Rd., Austin, Texas 78759. McKelley is an advisory programmer on the LAN NetView vendor support team. In his twenty-six years with IBM he has held various technical and management positions in field support and software development.

Jacob M. Rzepka, IBM Corp., internal zip 9154, 11400 Burnet Rd., Austin, Texas 78759. Rzepka has worked for IBM since 1974. He currently provides technical support to developers using LAN Netview and has managed and developed communication products for 3270 emulation, the IBM ES 1.0 Database Manager product, and evaluation and testing of microprocessor-based optical and computer systems. Rzepka received a B.S. in electrical engineering from Northwestern University and an M.S. in computer science and control systems from the University of Minnesota. He has also completed postgraduate studies in information systems and electrical engineering at the University of Texas, Austin.

REFERENCES

IBM LAN NetView Agents Extended Communications Manager Managed Objects Catalog (IBM Doc. S96F-8497)

IBM LAN NetView Agents Extended Database Manager Managed Object Catalog (IBM Doc. SF96F-8496)

IBM LAN NetView Agents Extended LAN Server Managed Objects Catalog (IBM Doc. S96F-8498)

IBM LAN NetView Operating System Agents Managed Object Catalog (IBM Doc. S96F-8495)

X/Open Systems Management Management Protocols API (XMP) (IBM Doc. GC31-7049-00)

X/Open OSI-Abstract-Data Manipulation API (XOM) (IBM Doc. GC31-7048-00)



OS/2 DEVELOPER BACK ISSUES

LIMITED NUMBER OF ISSUES AVAILABLE

The following Back issues of OS/2 DEVELOPER are available for \$9.95 each. (Add \$2.50 for shipping)

YES, PLEASE SEND ME:

- ☐ Winter 1990, DOS to OS/2 Conversion
- ☐ Fall 1990, Multimedia & Graphics
- ☐ Fall 1991, Computer Integrated Manufacturing
- ☒ Winter 1992 **SOLD OUT**
- ☐ Spring 1992, 32-bit Tools
- ☒ Summer 1992 **SOLD OUT**
- ☒ Fall 1992 **SOLD OUT**
- ☐ Winter 1993, Migration Strategies
- ☒ Spring 1993 **SOLD OUT**
- ☐ July/August 1993, Taligent Spotlight
- ☐ Sept/Oct 1993, Networking with OS/2
- ☐ Nov/Dec 1993, SOM/DSOM

Subscribe to OS/2 DEVELOPER for one year and save 33% off the cover price.

YES, PLEASE SEND ME:

- ☐ 1 year (six issues) – \$39.95 in the U.S.

Canada, Mexico, and international surface mail—add \$16 per year
International air mail—add \$30 per year

Please check the appropriate boxes on this page and complete the form below, and mail this page to:

OS/2 DEVELOPER

P.O. Box 1079
Skokie, IL 60076-9772
or fax: 708-647-0537
phone: 800-926-8672 (800-WANT-OS2)

Please Print:

NAME _____

TITLE _____

COMPANY _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

- ☐ Check enclosed
- ☐ Charge my credit card: ☐ Visa
- ☐ Mastercard
- ☐ American Express

CARD NUMBER _____

EXPIRATION DATE _____

SIGNATURE _____



New Tools, Applications, And Support



TOOLS

CICS Wrapper for PARTS Workbench. A component for the PARTS Workbench, Digitalk's CICS Wrapper allows developers to connect PARTS applications to on-line transaction processing power. PARTS is a client/server integration toolset that enables visual application construction from prefabricated software components. Developers can use the CICS Wrapper to access mainframe transactions and data.

Digitalk Inc.

Phone: (310) 645-1082 Fax: (310) 645-1306

DeskMan/2. DeskMan/2 manages the OS/2 Workplace Shell, selectively saving, restoring, and migrating WPS objects, controlling object styles, deleting undeletable objects, generating REXX scripts, managing icons, and so on. A SOM extension to the Workplace Shell, DeskMan/2 provides both GUI and programmatic interfaces, is CID enabled, and allows administrators to restrict access to individual DeskMan/2 features on a user-by-user basis.

Development Technologies Inc.

Phone: (803) 790-9230

Distributed Application/2. IBM's Distributed Application/2 (DA/2) is a set of APIs that provide a consistent way to access interprocess and network communication functions under OS/2 2.0, making client/server applications easy to create. The APIs hide the complexity of the supported communications protocols (APPC, NetBIOS, and OS/2 named pipes) and enable protocol selection at run time without requiring code changes.

IBM Corp.

Phone: (800) 342-6672

HALO Imaging Library. Media Cybernetics' HALO Imaging Library (HIL) offers developers the ability to add color, grayscale, and binary images to 32-bit applications quickly and easily. HIL provides more than 100 imaging functions and commands, enabling programmers to develop C programs that can read and store image files in several file formats and perform sophisticated processing of images in memory. A single HIL OS/2 workstation license costs \$599.00.

Media Cybernetics Inc.

Phone: (301) 495-3305 Fax: (301) 495-5964

KIShell. A workflow process modeling and enactment tool that takes advantage of modeling techniques like IDEF, UES Computing Environments' KIShell is the first commercial process management tool that can declaratively model work activities and synchronization, user responsibilities, and information, and tool use in a reengineered business process. KI Shell also enhances software engineering environments by integrating methodology process support, software project and structure information, and multi-vendor CASE tools.

UES Corp.

Phone: (614) 792-9993 Fax: (614) 792-0998

XPRO 5.0. A Prolog system for building intelligent applications that run in the 32-bit OS/2 environment, Rational Visions' XPRO 5.0 is designed to work with developers' C or C++ code. Consisting of the XPRO Developer, the XPRO Engine, and the XPRO Rule Compiler, the system will cost \$399 until December 31, 1993; after that \$499.

Rational Visions

Phone: (602) 846-0371



APPLICATIONS

C-Kermit. Columbia University's C-Kermit is a full-function communications software package for OS/2. With a built-in VT102 terminal emulator with screen, screen rollback, and color and printer control, C-Kermit allows easy switching among applications. Distribution fees range from \$45.00 to \$220.00; call for a catalogue.

Kermit Distribution, Columbia University
Phone (212) 854-3703 Fax (212) 662-6442

FAR Fax. A multi-application fax platform from Far Systems, FAR Fax is a stand-alone fax server for automating functions such as fax broadcasting or retrieval. FAR Fax supports Brooktrout fax boards.

Far Systems Inc.
Phone: (414) 563-2221 Fax: (414) 563-1865

IBM ARTIC960. IBM's first 32-bit communications coprocessor adapter card for Micro Channel-compatible systems such as the IBM PS/2 enables computers to communicate much more quickly. ARTIC offloads compute-intensive tasks, resulting in increased communications capabilities and improved performance. Cards start at \$2,200.

IBM Corp. Phone: (800) 342-6672

IBM X Window System Client Kit and OS/Windows Access. These two new products from IBM let users run X Window System applications and DOS or Windows TCP/IP applications, respectively, on OS/2. The X Window kit also lets applications run from OS/2 or another X Window server, while OS/Windows Access can also run applications written for DOS 2.1 in an OS/2 DOS session.

IBM Corp. Phone: (800) 342-6672



DEVELOPER SUPPORT

OS/2 Screen Saver Contest. BocaSoft, The IBM Developer Connection CD, and OS/2 Developer announce the 1993 OS/2 Screen Saver Developer Contest. Create an OS/2 screen saver module and win a \$1000 grand prize, OS/2 software products, T-shirts, and more. Entries will be accepted in three categories:

- Programmer—Submit working code for a screen saver module using the BocaSoft Wipe-Out Developer kit.
- Graphic Artist—Submit a screen saver storyboard with computer artwork included.
- Video Artist—Submit a video to be converted into an Ultimotion movie as a screen saver module.

Entries will be judged on appearance, originality, and functionality. BocaSoft will award prizes to the top three entries in each category. The grand prize and category winners will be selected by a panel of industry experts and announced at the Software Development Conference in March 1994. Contest results will be published in a future issue of OS/2 Developer.

Details and an official entry form are available on CompuServe in the BocaSoft section of the IBM OS/2 Vendor Forum in CONTEST.TXT. For addi-

tional information, contact BocaSoft at 71333,3617.

Mail entries with entry forms to:
1993 OS/2 Screen Saver Contest
117 NW 43rd St.
Boca Raton, Fla. 33431

Entries must be received before February 28, 1994.

IBM AskPSP Expert Systems CD-ROM. IBM Personal Software Products' AskPSP is an expert systems application that uses intelligent retrieval technology to provide OS/2 developers with answers to technical questions and problems. AskPSP, available free as part of a pilot program, provides an on-line "help desk" that responds to natural language description of problems with answers culled from a library of case histories or additional questions that refine the search.

For a free copy of Ask PSP, call IBM PSP Developer Assistance at (407) 982-6408.

To Appear In Product Watch

If you would like your product to appear in a future Product Watch, please fax a short press release to OS/2 Developer Product Watch, (415) 905-2234.

Press releases will be edited. Space is limited; OS/2 Developer cannot guarantee the inclusion of any product in these pages.



Software Tools

Software Installer for OS/2 lets you create CID-enabled LAN installations, shrink-wrapped disk or CD-ROM installations, or installations from a mainframe. **BY KEN GREENLEE, RUSSELL SAGRAVES, RUTH WILLENBORG, and BRIAN YOUNG**

Software Installer: Installations Made Easy

You've written your application and now you need an installation program. At a minimum, your customer is expecting an easy-to-use program that provides features such as allowing a choice of where to install your product, checking for the required hard disk space, and updating the user environment. In addition, many customers now expect a choice of disk, CD-ROM, or LAN install. Users can also be frustrated if an install program does not provide an automated deletion to undo installation. In the LAN environment, customers may even want



(l-r) Russell Sagraves, Brian Young, Ruth Willenborg, Ken Greenlee

to install your application unattended from a centralized location.

Providing this level and variety of installation function is expensive and time-consuming. You would rather spend your time improving your application's function, but because an installation program is a user's first impression of your product, meeting user expectations is critical.

SOFTWARE INSTALLER FOR OS/2

Software Installer for OS/2 addresses user expectations about installation. For example, the unattended LAN install is supported through compliance with IBM's Configuration, Installation, and Distribution (CID) architecture. The CID architecture requires each application to support the following requirements:

- Transfer of product to a repository

- Support of response files
- Support of command line parameters
- Generation of error and history logs
- Support for return codes to a Software Distribution Manager (SDM).

CREATING YOUR INSTALLATION PROGRAM

Once you've decided to use Software Installer, you need to enable your product. Users won't know that you are using Software Installer because your customized installation becomes embedded within the finished product. You can also provide the Software Installer to customers; they can use it later to restore, delete, and apply fixes to your application.

Before beginning the enablement process, you need to make decisions about how your product will be packaged and how you want to configure your customer's environment:

1. On which media (CD-ROM, disk, LAN, or host) will my product be delivered?
2. Will my product be packaged as one product or separate products?
3. Does my product have components? (A component is a portion of the product that the customer can optionally install.)
4. How many directories does my product require?
5. Does my product require changes to the customer's CONFIG.SYS or OS2.INI?
6. What Workplace Shell objects (such as folders or programs) does my product need?

As an exercise, we will take you through the steps



we went through when we enabled Software Installer to itself. We asked ourselves the same set of questions and arrived at the following answers:

1. Disk media
2. One product
3. Consists of four components (base install program, online reference, sample enablement files, and disk generation utility)
4. One directory
5. No CONFIG.SYS/OS2.INI changes
6. One main folder on the desktop containing program objects for the install program and related utilities, the on-line reference, samples, and the disk generator.

Now we are ready to begin the enablement process.

ENABLEMENT STEPS

Enabling to Software Installer takes six basic steps:

1. Rename Software Installer product files
2. Customize the initial screen
3. Build a customized copy of Software Installer
4. Create catalogue, package, and description files
5. Create your product files
6. Package your product.

A full description of these steps can be found in the online reference provided with Software Installer.

For our exercise, the enablement steps are shown in Figure 1. Let's look at each of them.

RENAME SOFTWARE INSTALLER PRODUCT FILES

Most Software Installer product files are shipped with a prefix of EPF; before you ship Software Installer with your product you must rename the files. Using the ISREN.CMD rename utility shipped with Software Installer, supply your own three character prefix and a directory for the renamed files.

CUSTOMIZE THE INITIAL SCREEN

This step is probably the most fun. You get to create bitmaps that are used on the initial Software Installer screen. You can use special effects to move the bitmaps on the screen and make them appear in slices or dissolves.

All screen customizations are done in a file called IIRC.RC. You will want to use the supplied IIRC.RC file as a starting point.

The customized screen used by Software Installer is shown in Figure 2.

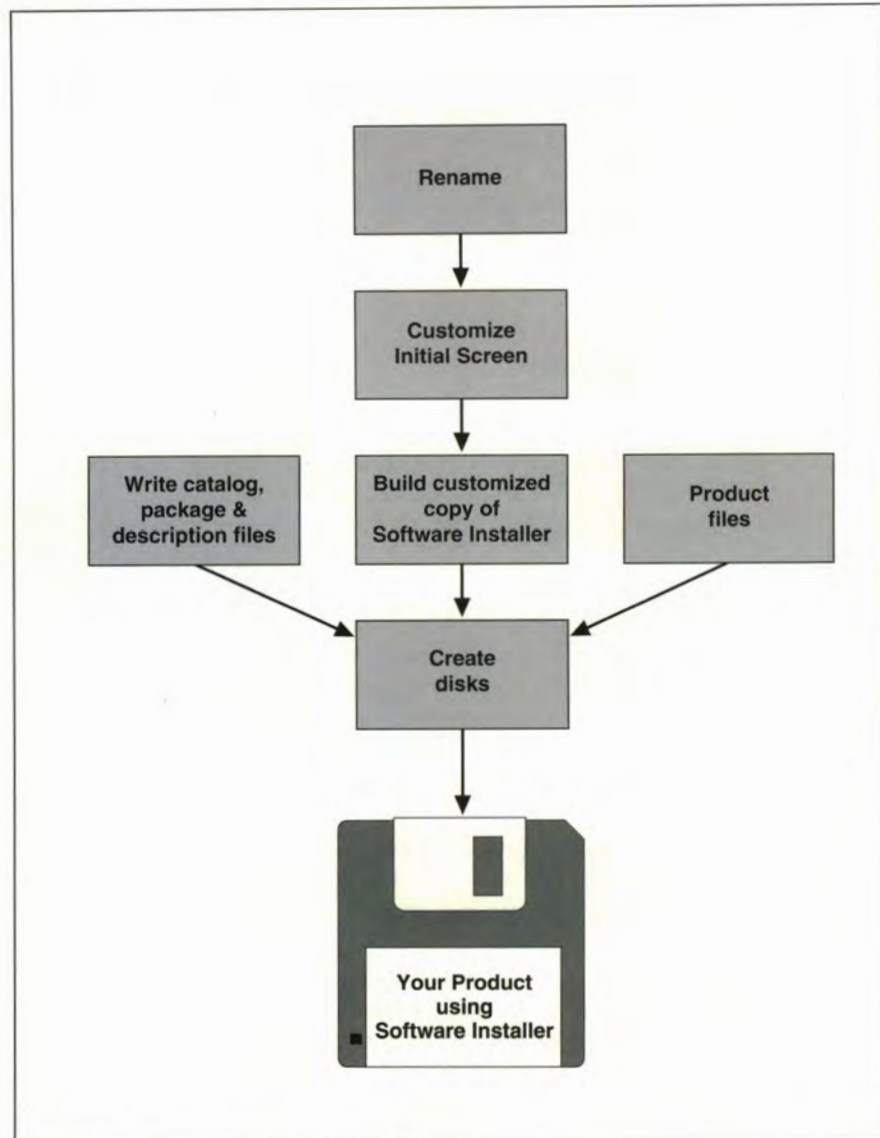


Figure 1. Steps for enabling to Software Installer

BUILD A CUSTOMIZED COPY OF SOFTWARE INSTALLER

This step builds your customized copy of Software Installer to ship with your product. You invoke the

supplied BLDINST.CMD utility to create a packed `INSTALL.IN_` file containing all of the customized files.

The important thing to remember is that before you invoke the BLDINST.CMD build utility, you need to be in the directory in which your renamed copy of Software Installer was created.

CREATE CATALOGUE, PACKAGE, AND DESCRIPTION FILES

This step will be the most time-consuming; you'll need to familiarize yourself with the terminology and syntax of the catalogue and package files. The terminology used in catalogue and package files is *entry type* and *keyword*.

The syntax of an entry type is:

```
entry type
  keyword1=value1,
  keyword2=value2,
  . . keywordn=valuen
```

Use entry types to select Software Installer features and keywords to define how the feature works.

Catalogue File. A catalogue file is a text file containing information about your product; this information is used by Software Installer to install your product. General product information such as name and version/release/modification number (VRM) must be provided for all products.

The answers you gave to the enablement questions will help you decide which additional entry types and keywords to use.

The answers we provided to the enablement questions and the corresponding catalogue file entries to use are shown below:

1. Disk—use the `DRIVE:` value keyword
2. One product—use one `PACKAGE` entry type.

As an example, a subset of the actual catalogue file is shown in Figure 5. For a complete list of catalogue file entry types, see Table 1.

Package File. A package file is a text file that contains information about your

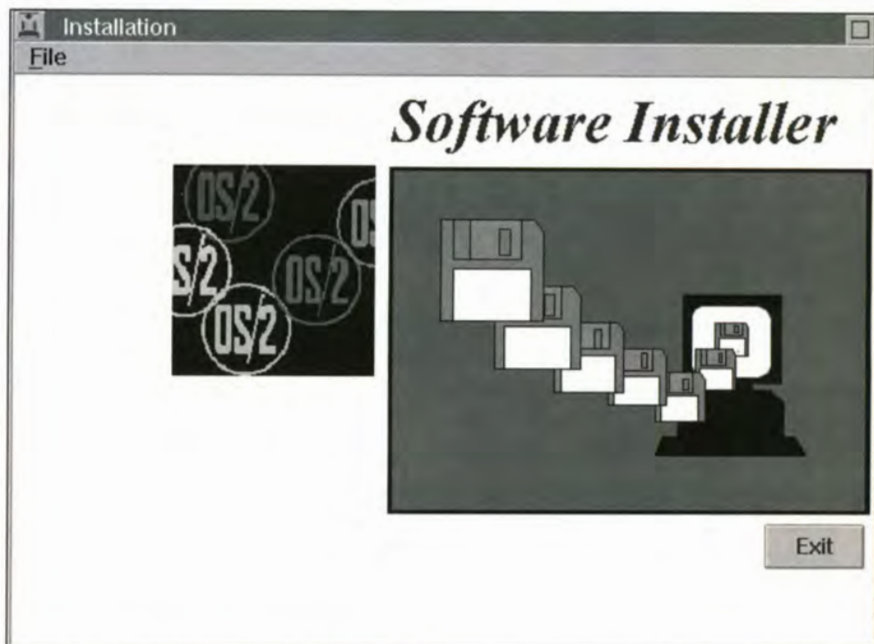


Figure 2. Software Installer's customized screen

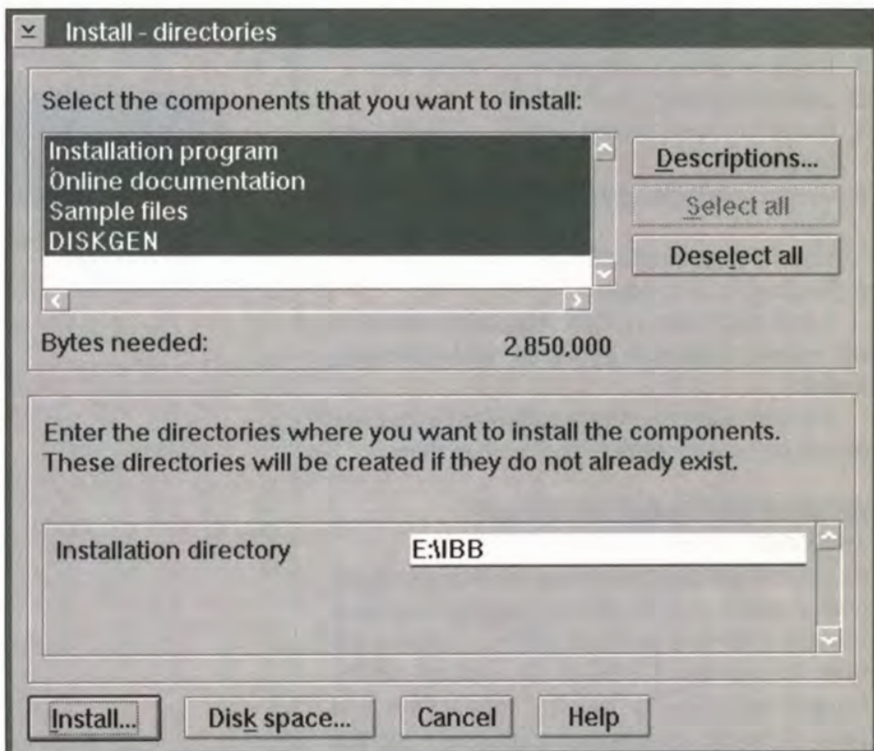


Figure 3. Software Installer's Install-directories dialog box

product files; this information is used by Software Installer to install your product and to customize your customer's workstation. General product information such as service level must be provided.

The answers you gave to the enable-

ment questions will help you decide which additional entry types and keywords to use.

The answers we provided to the enablement question and the corresponding package file entries to use are shown here:

1. Disk—use the DRIVE: value on the SOURCE keyword of the FILE. Entry type—use a DISK entry for the one disk we ship.
3. Yes—use four COMPONENT entries, one for each component.
4. One directory—our PATH entry contains only the FILE keyword.
6. One main folder—use the CRE-ATEWPSOBJECT and DELETEWPSOBJECT exits.

For example, a subset of our Software Installer package file is shown in Figure 6. For a complete list of Software Installer exits, see Table 2. For a complete list of package file entry types, see Table 3.

Description File. A description file is a text file that contains a brief description of the product. The text in the description file is displayed exactly as you type it. The only formatting that occurs is word wrapping.

For a simple disk installation, you do not need a description file.

COLLECT YOUR PRODUCT FILES

In this step, you should move all your product files to a central location for packaging.

PACKAGE YOUR PRODUCT

In this step you create a master copy of your product that will be shipped to customers. If you are shipping your product on disk, a disk generator utility is provided to place your files onto the disks.

If you decide to make your product available on both disk and CD-ROM, you will need two package files: one for the disk installation and one for the CD-ROM installation.

INSTALLING YOUR PRODUCT

To start an installation from a disk, insert the disk into the A: drive and enter A:\INSTALL. Your installation will display your customized screen, prompt the user whether to update his or her CONFIG.SYS, prompt the user to select the components, display a progress indicator as the component files are installed, and display a mes-

sage when the installation is complete.

See Figure 3 for a picture of Software Installer's Directories dialogue. See Figure 4 for a picture of the Progress dialogue.

LAN AND HOST SUPPORT

For the LAN environment, both repository and server/requester scenarios are supported. The repository scenario allows the application to be installed on



USE IT

GammaTech Utilities or for OS/2

The **GammaTech Utilities** are designed to enhance and complement your OS/2 system. These utilities provide the ability to perform vital maintenance and recovery operations easily without extensive technical knowledge. Use **GammaTech Utilities** for your critical data or you might lose it.

- Undelete Erased Files
- Optimize HPFS and FAT Volumes
- Backup INI and Desktop Files
- Recover HPFS Volumes
- Protect and Backup Boot Sectors
- Identify and Mark Bad Sectors
- Reconstruct Boot Sectors
- Protect/Lock Files
- Permanently Erase Sensitive Data
- Mass Delete Selected Files
- Sort FAT Directories
- View and Edit Selected Files
- Disk Sector Edit (ASCII or Hex)
- Display Volume Information
- Alter File Attributes
- Add Comments to Files
- Display File Fragmentation
- Display Directory Information
- SAA/CUA Compliant PM Interface

For OS/2 Version 2

VISA, MasterCard, American Express, C.O.D. • Overnight Service Available

(405) 947-8080 • Fax (405) 632-6537

SofTouch Systems, Inc., Workstation Division

1300 S Meridian, Suite 600, Oklahoma City, Oklahoma 73108

a LAN server that is the source of the install for all workstations that have access to the server. In this scenario, the entire application would be transferred to each requester that initiates an install.

The server/requester scenario allows a part of the application to be installed on a server and then shared by all requesters. This part only exists on the LAN server and is never installed onto the requesters. A separate part of the application that can't be shared is installed on each requester. Depending upon how the application uses Software Installer, this piece can be installed on each requester from the server or from one of the other media, such as a CD-ROM.

For the host environment, Software Installer supports installation from a system running one of the following operating systems:

- Multiple Virtual Storage (MVS)
- Virtual Machine (VM)
- Operating System/400 (OS/400) with IBM PC Support/400
- Virtual Storage Extended (VSE/ESA) 1.3.0.

ADVANCED TOPICS

Maintenance Functions. Software Installer includes maintenance functions for updating, restoring, and deleting. You do not need to make any special modifications to your catalogue or package file to access these functions; they may be accessed through either a command line parameter or a Software Installer dialogue.

User Exits. You may be thinking that your product installation is too specific to be handled by a generic installation package. Perhaps you bring up dialogues at various points of your current installation to prompt for information about the user's system or the user's choice of configuration options. Then you use this input to finish installation and configuration of the product.

Software Installer is capable of handling even the most unique installation processes. This is done by allowing you to execute your own exits during instal-

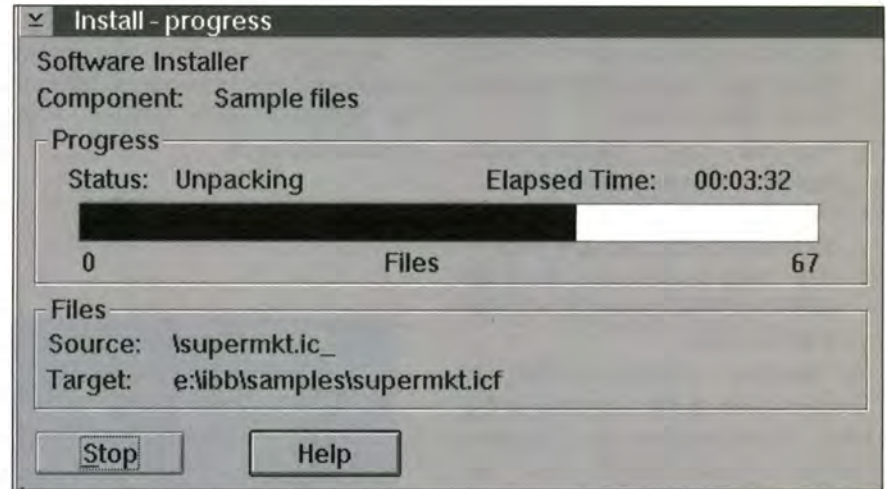


Figure 4. Software Installer's progress dialogue

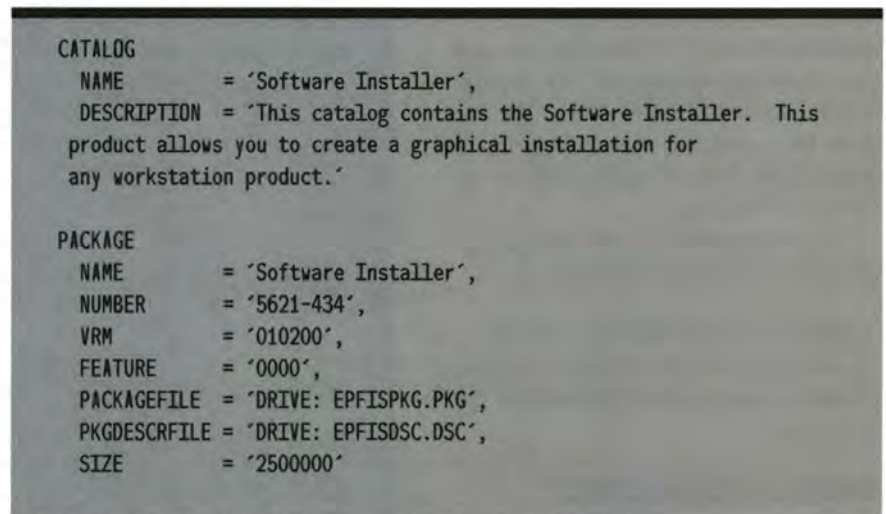


Figure 5. Subset of Software Installer's catalogue file

lation. You can code short exits to perform tasks, such as updating configuration files used by your product, or bring up dialogues to prompt for additional information. For example, you could code an exit to bring up a dialogue to prompt for the type of printer the user has.

Inter-Exit Communication. Now that your user exit has obtained the customer information you needed to continue with the install, how do you reference this information?

Exits can communicate back to Software Installer and, as a result, to later exits during product installation. Software Installer provides APIs that enable your exits to store and retrieve

values for variables you wish to set. By referencing the values of these variables in the package file, you can determine whether a file is to be transferred, a configuration file updated, or any other action taken. For example, you may wish to transfer a certain printer driver only if the user selected the printer in a previous user exit's dialogue. By setting a variable to the name of the printer selected, you can reference the variable later in a FILE entry type to determine whether the file should be transferred.

CONFIG.SYS Manipulation. Software Installer allows advanced manipulation of the CONFIG.SYS file. Using the ADDCONFIG and UPDATECONFIG entry types, you


```

SERVICELEVEL
  LEVEL = '000000'

DISK
  NAME = 'Software Installer - Diskette 1',
  VOLUME = 'IBBENU01'

PATH
  FILE = 'C: /IBB',
  FILELABEL = 'Installation directory'

* Product Folder
COMPONENT
  NAME = 'Product Folder',
  ID = 'PRODFLD',
  DISPLAY = 'NO',
  SIZE = '10000'

* Create the folder
FILE
  EXITWHEN = 'INSTALL || UPDATE || RESTORE',
  EXITIGNOREERR = 'NO',
  EXIT = 'CREATEWPSOBJECT WPFolder "Software Installer"
  <WP_DESKTOP> R
  "ICONFILE=%EPFIFILEDIR%\bin\epfiico1.ico;
  OBJECTID=<EPFIFLD>"

* Installation Program
COMPONENT
  NAME = 'Installation program',
  ID = 'INSPROG',
  REQUIRES = 'PRODFLD',
  DISPLAY = 'YES',
  DESCRIPTION = 'Select the "Installation program" component to
  install all files of Software Installer that will allow you to
  create a renamed copy of Software Installer and to perform the
  installation, maintenance, and delete functions of Software
  Installer.',
  SIZE = '2120000'

* Register the Installation Utility
FILE
  EXITWHEN = 'INSTALL || UPDATE || RESTORE',
  EXITIGNOREERR = 'NO',
  EXIT = 'CREATEWPSOBJECT WPProgram "Installation Utility"
  <EPFIFLD> R
  "EXENAME=CMD.EXE;OBJECTID=<EPFINSTS>;
  STARTUPDIR=%EPFIFILEDIR%\bin;
  PARAMETERS=/c %EPFIFILEDIR%\bin\epfinsts.exe;"

FILE
  EXITWHEN = 'DELETE',
  EXITIGNOREERR = 'YES',
  EXIT = 'DELETEWPSOBJECT <EPFINSTS>'

```

Figure 6. Subset of Software Installer's package file (continued on page 111)

Footprints



for OS/2

- Tired of **slow** debuggers?
- Can't duplicate a reported customer problem?
- Need to verify all paths of execution without using *printf*?
- Trying to support code in Dime Box, Texas, from Brooklyn, New York?

Footprints is the product for you!

Dynamic debugging
Online monitoring/Offline analysis
Realtime systems
Telecommunications
Multi-thread / multi-process
Remote support and postmortems

Dynamically select which portions of code to trace - in a live user environment!

Footprints is simple to use! Just include a header. Link to a DLL. You're ready to call **Footprints** and begin capturing any or all the data you require. **Footprints** doesn't bog down your code with constant tracing or file I/O. *The tracing only gets activated when and where you want it.* From our friendly **PM interface**, you dynamically turn on one or more of up to 64 separate trace ids that control what data you are tracing. You can then view the time-stamped data in memory as it whizzes by, or view it at your leisure from a file days after the fact.

ONLY \$ 99

Call **713-871-1448** now!

VISA/MC or send a check to
Hardy Software Systems, Inc.
4801 Woodway, #415 • Houston, TX 77056
s/h charge \$ 5.95
Total Cost \$104.95
Texas residents add 8.25% sales tax.
Requires OS/2 2.0 or later.



Hardy Software Systems, Inc.

ADVERTISER INDEX

Company, Fax Number	Page	Company, Fax Number	Page	Company, Fax Number	Page
Borland International.	COV4 (408) 439-0115	IBM Personal Software Products.	10-11 (407) 982-6408	Parallel Storage Solutions	25 (914) 347-4646
Burton Systems Software.	69 (919) 233-0716	IBM Personal Software Products13, 15, 17, 19, 21	(407) 982-6408	Programmer's Paradise	4
CCT.	63 (612) 339-5965	IBM Personal Software Products	87 (407) 982-6408	Quadron	69 (805) 966-6424
Creative Systems Programming.	31 (609) 234-1920	IBM Santa Theresa Laboratories.	49	Real Time Technologies	63 (708) 446-6886
Digitalink Inc.	43 (310)645-1306	Ingres	57	Rhetorex Inc..	85 (408) 370-1171
Essex Systems Inc..	63 (508) 750-4699	Impact Software.	63 (818) 879-5593	Rimstar Technology.	23 (603) 778-2408
Gamma Tech	107 (405) 359-7391	Indelible Blue Inc.	63 (212) 629-8819	Sequiter Software.	COV3 (403) 448-0315
Globalink.	89 (703) 273-3866	Intersolv	45 (503) 645-4576	Softbridge	95 (617) 864-7747
Gpf Systems Inc..	80 (203) 873-2171	Kaseworks	37 (404) 448-4163	Technically Speaking	63 (508) 881-7320
Graphical Software Interfaces	25 (404) 382-6374	Levine Computer Consultants.	25 (703) 790-1660	Token Technology.	33 (415) 965-8658
Hardy Software System.	109 (713) 871-1449	Micro Focus	2 (415) 856-6134	Van Nostrand Reinhold	55 (606) 525-7778
HockWare	53 (919) 380-0757	Microformatic	73 (203) 648-9587	Walnut Creek CDROM.	63 (510) 674-0821
IBM Canada	65 (800) 445-2426	Mitnor Software.	79 (918) 357-2869	Watcom C for IBM PCs	COV1 (519) 747-4971
IBM GIK	35 011-43-1-21145-4490	Object Design	27	Zinc Software Inc.	29 (801) 785-8996



The **Impact Formatted Entryfields** are a completely new window class which allows the developer to specify a format string when the entryfield is created. Formatting can be used for fields like Phone Number, Dates, Social Security Number, etc.

The **Impact Formatted Entryfield** dynamically formats itself as the user inputs data.

The **Impact Formatted Entryfield** support assures uniformity in data entry.

The developer has complete control to create custom fields like;

- **Numeric entry only**
- **Alpha characters only**
- **Special delimiting characters**
- **Value ranges**
- **Currency support** and, much more.

Easy to use! Just change the window class in your resource file.

Download free demo of Impact Entryfields from our BBS.
(818) 879-7405

Add Impact to your application



5889 Kanan Rd.
Suite 330
Agoura Hills
CA 91301

To order call or write
(800) 676-9390
(818) 879 5592
FAX (818) 879-5593

Invest a stamp



Save a bundle

For the price of a stamp, you can get the latest edition of the federal government's free **Consumer Information Catalog** listing more than 200 free or low-cost government publications on topics such as federal benefits, jobs, health, housing, education, cars, and much more. Our booklets will help you save money, make money, and spend it a little more wisely.

So stamp out ignorance, and write today for the latest free **Catalog**. Send your name and address to:

Consumer Information Center
Department SB
Pueblo, Colorado 81009



```
FILE
  UNPACK = 'YES',
  SOURCE = 'DRIVE: VARS.OB_',

  VOLUME = 'IBBENU01',
  PWS = 'bin/vars.obj'

* OnLine Documentation
COMPONENT
  NAME = 'Online documentation',
  ID = 'INSDOC',
  REQUIRES = 'PRODFLD',
  DISPLAY = 'YES',
  DESCRIPTION = 'Select the "Online documentation" component to
install the Software Installer Reference, giving you all of the
information you will need to use Software Installer.',
  SIZE = '360000'

* Samples files
COMPONENT
  NAME = 'Sample files',
  ID = 'INSSAMP',
  REQUIRES = 'PRODFLD INSPROG',
  DISPLAY = 'YES',
  DESCRIPTION = 'Select the "Sample files" component to install the
sample catalog, package, and description files. These files show
you how to use most of the features of Software Installer.',
  SIZE = '120000'

* Disk generation utility
COMPONENT
  NAME = 'Diskette Generator',
  ID = 'DSKGEN',
  DISPLAY = 'YES',
  DESCRIPTION = 'Select the "Diskette generator" component to install a
utility that will create diskettes for distribution or upload
your product files to a host system for distribution.',
  SIZE = '140000'
```

Figure 6. Subset of Software Installer's package file (continued from page 109)

CATALOG	Provides a name and description of what the catalogue file contains
INCLUDE	Specifies an additional catalogue file to include into the primary catalogue file
PACKAGE	Specifies the product name and numbers, as well as the names of the package and description files

Table 1. Catalogue file entry types

may add a new line or update an already existing line. You may position your new item before or after another item that you specify. You may also replace the first, last, or all occurrences of a line with a new line. You can ensure that a new line is added only if it does not exist already. Finally, should the user delete your product, the CONFIG.SYS updates can be undone.

User-Supplied Pack Utility. Files can be packed with either the supplied pack utility or a utility of your choice. You need only supply in the package file or Disk Generation Utility the name of the utility you wish to use in place of the default.

CONCLUSION

Software Installer for OS/2 provides a rich, functional installation and maintenance program for applications. Your customers will get a great first impression of your product, and you can spend your time enhancing your product.

REFERENCES

For detailed information on creating your installation programs, please refer to the *Software Installer for OS/2 Reference* and the product samples.

Software Installer for OS/2 is currently available in English and Japanese. For more information, call (800) 426-2279 (919-469-7763) or fax 919-469-7423.

Questions are also addressed on the OS/2 Developer forum (OS2DF2) on CompuServe section 8 and the INSTALL2 forum on the OS/2 Talklink BBS.



Exit	Function
ADDINI	Add application name/keyname pairs or modify information in the OS2.INI system file.
ADDPFILE	Add application name/keyname pairs or modify information in a specified application profile file.
CREATEWPSOBJECT	Create a workplace object, such as a folder.
DELETEFILES	Delete data files your application created.
DELETEINI	Delete application names from the OS2.INI system file.
DELETEPROFILE	Delete application names from a specified application profile file.
DELETEWPSOBJECT	Delete a workplace object.
DEREGISTERWPSCLASS	Deregister an OS/2 object class.
EXEC	Start a user exit. This exit may be a program (.EXE) or command file (.CMD).
REGISTERWPSCLASS	Register an OS/2 object class.
SETVAR	Set installation variable for use later in the package file or in user exits.

Table 2. Software Installer exits

Ken Greenlee, IBM Software Solutions, P.O. Box 60000, Cary, NC 27512. Greenlee is a senior associate programmer with IBM and the team leader for Software Installer. Since joining IBM in 1989, he has been a software developer for several host and workstation products. He holds a B.S. in computer science from Indiana University.

Russell Sagraves, IBM Software Solutions, P.O. Box 60000, Cary, NC 27512. Sagraves is a senior associate programmer with IBM Application Platform Products, currently working on NLS for the Software Installer product. Since joining IBM in 1980, he has worked as a customer engineer, marketing support representative, and programmer. He holds a B.S. in mechanical engineering from North Carolina State University.

Ruth Willenborg, IBM Software Solutions, P.O. Box 60000, Cary, NC 27512. Willenborg is the development manager responsible for the Software Installer product. Since joining IBM in 1985, she has worked on the FAA Advanced Automation System and in a variety of development, architecture, and management positions. She holds a B.S. in computer science from Duke University.

ADDCONFIG	Specifies lines to be added to, deleted from, or replaced in the user's CONFIG.SYS file
COMPONENT	Identifies a separately installable part of the application
DISK	Associates a user-friendly name to an installation disk
EXIT	Specifies the DLL to be used for installation exits
FILE	Identifies files to be installed and exits to be run
INCLUDE	Specifies another package file to be included into the current one
OPTIONS	Specifies message text to be displayed after a successful install, update, restore, or delete
PACKFILE	Identifies a packed file that contains multiple product files
PATH	Specifies the paths to be used for transferring all files
SERVICELEVEL	Specifies the level of service of the product files
UPDATECONFIG	Specifies information to be added to existing CONFIG.SYS lines

Table 3. Package file entry types

Brian Young, IBM Software Solutions, P.O. Box 60000, Cary, NC 27512. Young is a co-operative education student from North Carolina State University, where he is working on the final year of his B.S. in computer science.

In over a year of co-operative education with IBM, Young has worked on the Workstation Platform, 32-bit conversion of Software Installer, and implementation of the expanded CONFIG.SYS handling of Software Installer.

CodeBase 5.0

New for OS/2

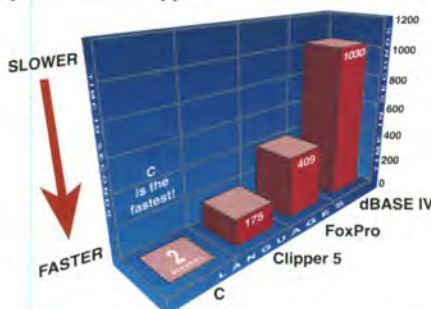
Discover why FoxPro, Clipper, and dBASE were all written in C.

There is a good reason why your database language was developed in C. In fact, there are many good reasons.

C code is small. C code is fast. C code is portable. C code is flexible. C is the language of choice for today's professional developer. With the growing complexity of database applications, C is a realistic alternative. Now with CodeBase 5.0, you can have all the functionality, simplicity and power of traditional database languages together with the benefits of C/C++.

C speed - fast code, true executables...

FoxPro, Clipper, and dBASE were written in C primarily for speed. But those compilers don't really compile, they combine imbedded language interpreters into your .EXE. Now that's slow. For dazzling performance you need the true executables of C. With CodeBase you get the real thing, C code. Consider the following statistics, from the publisher of Clipper:



"Sieve of Erastosthenes"
Benchmark for Prime Number Generation
Shows C to be incredibly faster!

C size - small executables, no added overhead...

FoxPro, Clipper and dBASE would like you to believe you need their entire development system to build database applications. But

remember, those products are all written in C. So why do you need to lug all their extra code around? You don't. CodeBase is a complete DBMS, in C. No fat executables stuffed with unused code. No runtime modules. No royalties. Just quality C code. CodeBase is just what you need.

C portability - ANSI C/C++ on every hardware platform...

No other language exists on more platforms than C/C++. Why rewrite your entire application for DOS, Windows, Windows NT, OS/2 or UNIX? With CodeBase the complete C source code is included, so you can port to any platform with an ANSI C or C++ compiler. Now and in the future.

dBASE Compatible data, index and memo files...

You want the industry standard. You need compatibility. Sure, dBASE is the standard, but every dBASE compatible DBMS product uses its own unique index and memo file formats. Only CodeBase has them all: FoxPro (.cdx), Clipper (.ntx), dBASE IV (.mdx) and dBASE III (.ndx). Now it's your choice, we're compatible with you.

Announcing CodeBase 5.0

The power of a complete DBMS, the benefits of C

NEW - Multi-user sharing with FoxPro, Clipper and dBASE...

Now your multi-user C/C++ programs can share data, index and memo files at the same time as concurrently running FoxPro, Clipper and dBASE programs. No incompatibilities. No waiting.

NEW - Queries & Relations 1000 times faster...

CodeBase 5.0 now lets you query related

data files with any logical dBASE expression. Our new Bit Optimization Technology (similar to FoxPro's Rushmore technology) uses index files to return a query on a 1/2 million record data file in just a second. Automatically take advantage of this query performance by using our new CodeReporter:

To use CodeReporter, simply draw your report, then include it in any program you write. Call 403/437-2410 now for your FREE working model of CodeReporter.

New - Design complex reports in just minutes...

Our new CodeReporter takes the painstaking work out of reports. Now simply design and draw reports interactively under Windows 3.1, then print or display them from any DOS, Windows or UNIX application.

SPECIAL - FREE CodeReporter

Order CodeBase 5 before June 30, 1993 and receive CodeReporter for free! This offer includes our no-risk, 90-day money back guarantee, so order today!



CodeBase 5.0
The C/C++ Library for DataBase Management

Call Now
403-437-2410

SEQUIER SOFTWARE INC. FAX 403-436-2999
Europe 33.20.24.20.14
#209.9644-54 AVE., EDMONTON, AB, CANADA T6E-5V1

Demand for breakthrough, 32-bit OS/2® applications is developing into a huge opportunity.

That's why we developed Borland® C++ for OS/2®—the easiest-to-use C++ development system for world-class OS/2 applications.

Borland C++ for OS/2 is a complete application development system. It combines an industry standard-setting Borland C++ compiler and

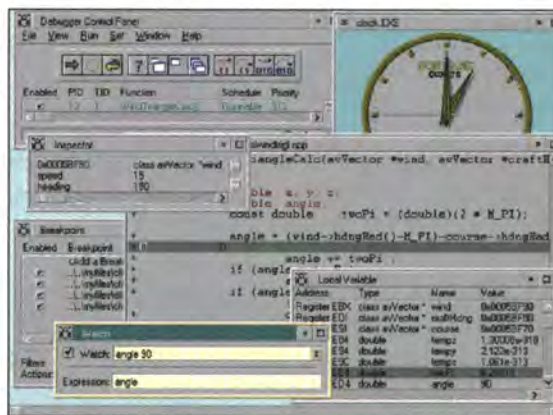
More exciting developments

tools with the industrial-strength power of OS/2, so building C and C++ OS/2 applications is faster and easier than ever before. Borland's highly integrated graphical environment and visual tools raise productivity to a whole new level. The new, integrated Turbo Debugger GX finds bugs fast. And you can take full

advantage of the full power of OS/2 with 32-bit code generation,

pre-emptive multitasking and state-of-the-art optimizations.

To help speed your development along, Borland C++ gives you access to the largest resources of OS/2 libraries, controls and other add-ins. So if you're looking for fast, easy ways to create exciting, new, 32-bit OS/2 applications, get Borland C++ for OS/2



Borland C++ for OS/2 has everything you need to succeed in creating leading-edge OS/2 applications.

today—and see what develops. To order or to find out more about OS/2 2.1 or Borland C++ for OS/2, call 1 800 3-IBM-OS2. In Canada, call 1 800 465-7999.

Operate at a higher level.™

Borland C++

IBM and OS/2 are registered trademarks and "Operate at a higher level" is a trademark of International Business Machines Corporation. Borland is a registered trademark of Borland International, Inc. ©1993 IBM Corp.

